UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INFORMÁTICA

Proyecto Fin de Carrera

SISTEMA DE AUTENTICACIÓN GLOBAL EN UNA RED EMPRESARIAL

Autor: Ignacio Jiménez Pinto
Tutor: Daniel Calzada del Fresno
Fecha: Junio 2014

Me gustaría dedicar este texto a mi familia, por su apoyo incondicional, y a mi profesor y tutor, Daniel Calzada, por su paciencia, su ayuda, y la formación recibida.

—Ignacio

ÍNDICE

1. INTRODUCCIÓN

1. Introducción

Actualmente las redes de datos están muy presentes en nuestra vida diaria. Hacemos uso de conexiones de red en gran parte de nuestra actividad personal y laboral. Con el uso y la expansión de la telefonía móvil de tercera generación la interacción es todavía mayor, y va en aumento. Esto significa que nuestra información personal y corporativa está cada vez más expuesta a la indiscreción y el uso fraudulento.

Los mecanismos de seguridad que ofrece la tecnología actual son muy seguros y eficientes, cuando son aplicados en su totalidad y de una manera correcta. Uno de los mayores inconvenientes de la seguridad suele ser la incomodidad que supone al usuario su aplicación, y sobre todo su mantenimiento. Para mejorar esta relación entre comodidad y seguridad se han desarrollado las actuales tarjetas de identificación electrónica (Smart Card), como el DNIe, con la misma finalidad que tuvieron en su momento las tarjetas de crédito (a nivel monetario).

Esta tecnología puede ser combinada con algunos protocolos de red modernos, incrementando así el nivel de seguridad de las infraestructuras de red de grandes organizaciones o pequeños particulares, redundando en beneficio de ambas partes (usuarios y organizaciones).

En una entidad corporativa actual, es habitual encontrar sistemas de control de acceso basados en credenciales simples, y particularizados para cada recurso en cuestión. El tipo de credencial más extendido, pero no por ello el más seguro (al igual que ocurre con algunos sistemas operativos), es el basado en parejas de valores *username/password*. Este tipo de

credencial posibilita el acceso a recursos variados y dispares dentro de una organización, como puede ser el acceso al correo electrónico corporativo, los servicios contratados por clientes de la entidad, la gestión y tramitación de nóminas de empleados, e incluso el mantenimiento de cuentas financieras de la compañía.

En sistemas SCADA (Supervisory Control And Data Acquisition) o entornos gubernamentales, este tipo de control de acceso ya no es admisible (o no debería serlo), y ha quedado obsoleto. La seguridad en estos sistemas críticos debe basarse en el control de acceso a la red en sí misma (802.1X), en la certificación digital (PKI), y en los dispositivos criptográficos portables (Smart Card), al igual que en una vivienda particular la seguridad se instaura en la puerta de acceso a la misma (blindada o acorazada), y no en las puertas interiores que dan acceso a las habitaciones, las cuales normalmente son simples y débiles. Por extensión, es este tipo de sistema de control de acceso el que está en proceso de expansión, y difundiéndose por entidades públicas y privadas.

Los sistemas de control de acceso actuales deben restringir la conexión a la infraestructura de red de la organización, haciendo uso de los elementos mencionados, y pudiendo además implementar diversas políticas de control, individuales o grupales, para permitir acceso personalizado a recursos internos, evitando así subdelegar el control de acceso para cada recurso en cuestión, o dicho de otra forma, deben establecer un control de acceso más seguro y más centralizado, de lo cual se encargan los sub-sistemas de autenticación.

Los empleados (extensible a clientes y proveedores) de la compañía deben autenticarse con una tarjeta de identificación electrónica (que contiene credenciales digitales), proporcionada por la entidad, para poder acceder a la red interna/externa de la compañía. El proceso es totalmente transparente para el empleado, y el procedimiento idéntico por cualquiera de los canales habilitados para el acceso (conexión cableada, inalámbrica, o remota).

Todos estos elementos, bien integrados e implantados, proporcionan una solución de acceso robusta y eficiente, basada en métodos de autenticación seguros, o dicho de otra forma, un sistema seguro de control de acceso global.

2. OBJETIVOS

2. Objetivos

El presente documento pretende ser un texto didáctico e instar a profundizar en las tecnologías expuestas, al mismo tiempo que fomentar su uso en una actividad profesional cotidiana y en un entorno tecnológico adecuado.

La pretensión del proyecto es abarcar los fundamentos, el diseño, la configuración, y la puesta en funcionamiento, de una infraestructura de comunicaciones empresarial de escala media, que facilite el acceso a los recursos de la empresa a sus clientes y/o personal de la misma, mediante un sistema de control de acceso global basado en autenticación y autorización, y que implemente e integre al mismo tiempo, mecanismos avanzados de seguridad mediante el empleo de tarjetas inteligentes.

Los objetivos particulares para cada una de las dos partes beneficiarias de una implantación de estas características, son los siguientes:

Organización participante:

1. Desplegar una infraestructura de control de acceso a tres niveles, cableado, inalámbrico, y remoto, para cubrir las necesidades operativas de su personal, permitiendo que puedan realizar sus funciones desde cualquier ubicación, dentro y fuera de la sede corporativa.

2. Utilizar métodos seguros, estandarizados, y fácilmente accesibles, para implantar los controles de acceso al sistema, garantizando así la

seguridad de todas las partes implicadas en los procesos de autenticación y autorización.

3. Emplear tecnología disponible en el mercado y asequible económicamente para cualquier organización, manifestando una clara preferencia por lo productos basados en *software* libre, debido, entre otros aspectos, a la flexibilidad de sus licencias abiertas, en contraposición a las licencias privativas, las cuales son más restrictivas y con mayores costes.

4. Cumplir todas las normativas, estándares, y procedimientos de seguridad necesarios, para garantizar la confidencialidad de la información gestionada, y el control de acceso al medio, según las necesidades que la organización pueda tener de cara al correcto ejercicio de su actividad profesional.

Personal participante:

1. La relación comodidad/seguridad debe estar siempre presente en su máximo exponente, mejorando el desempeño de las funciones laborales. Tan solo un código PIN de cuatro dígitos debe ser recordado, con tres intentos antes del bloqueo.

2. Debe tenerse la certeza de estar conectado a una red segura, y disponer de medios para garantizar la confidencialidad de la información transmitida, debiendo esta ser considerada en su totalidad como información sensible de la organización, sin excepciones.

3. Debe permitirse el uso de perfiles distintos en función de los credenciales facilitados, permitiendo la integración funcional y operativa entre el personal de distintos equipos o departamentos.

4. Debe garantizarse la accesibilidad del sistema desde cualquiera de los puntos de acceso disponibles, y haciendo uso de cualquiera de los medios facilitados a tal fin por la organización.

5. El personal es responsable de sus credenciales de acceso al sistema, tanto a nivel físico como lógico, debiendo notificar a los responsables del servicio cualquier pérdida, robo, o deterioro de los mismos, los cuales, en el peor de los casos, serán reemplazados por otros nuevos de características similares.

3. ENTORNO TECNOLÓGICO

3. Entorno tecnológico

En este capítulo se introducen conceptos y estándares, que constituyen los cimientos del entorno tecnológico que da soporte a la infraestructura desplegada en el capítulo 5 (Realización del proyecto). Su comprensión es fundamental para poder afrontar con garantía de éxito la implantación y el mantenimiento de la infraestructura abordada en el proyecto.

3.1 Sistema criptográfico

En la actualidad existen dos técnicas distintas para la aplicación de criptosistemas:

- Criptografía simétrica/de clave secreta.
- Criptografía asimétrica/de clave pública.

La primera (cifra clásica) surge ante la necesidad de proteger la información intercambiada por medio de un canal inseguro, impidiendo la legibilidad de la misma en el supuesto caso de que sea visualizada por un tercero ajeno a la comunicación. Pero para que la aplicación de esta cifra tenga éxito, ambas partes en exclusiva (emisor y receptor) deben conocer la clave secreta, y así poder establecer un canal seguro. Este problema del intercambio o compartición de la clave secreta lo resuelve la segunda técnica (cifra moderna).

Ambas cifras aplicadas conjunta y debidamente dan lugar a un criptosistema eficiente y muy seguro, pero igualmente pueden ser aplicadas por separado, pues son independientes.

A estas dos técnicas se suman también otros elementos comunes, que son necesarios para la funcionalidad del criptosistema, como son las funciones HASH, HMAC y PRF, las cuales permiten obtener un resultado exclusivo ante una entrada determinada, impidiendo la reversibilidad del proceso. Se utilizan para garantizar la integridad de los mensajes intercambiados.

El proceso de aplicación de una cifra clásica responde al mostrado en la siguiente figura:

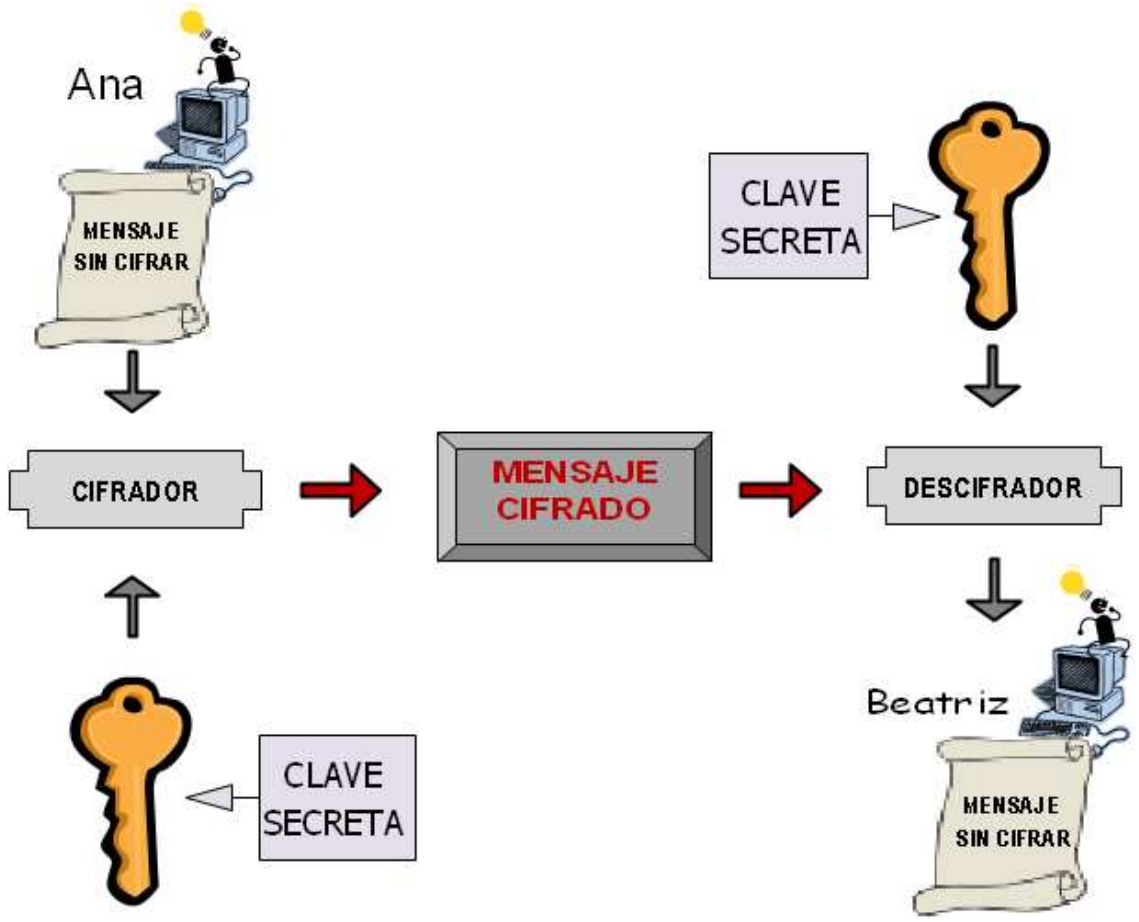

Figura 1.- Criptografía Simétrica.

El mecanismo de la figura 1 es bastante intuitivo, el mensaje se cifra en origen con la misma clave que se descifra en destino, y de ahí su simetría.

El resultado de aplicar la cifra al texto en claro se conoce con el nombre de criptograma.

El proceso de aplicación de una cifra moderna responde al mostrado en la siguiente figura:

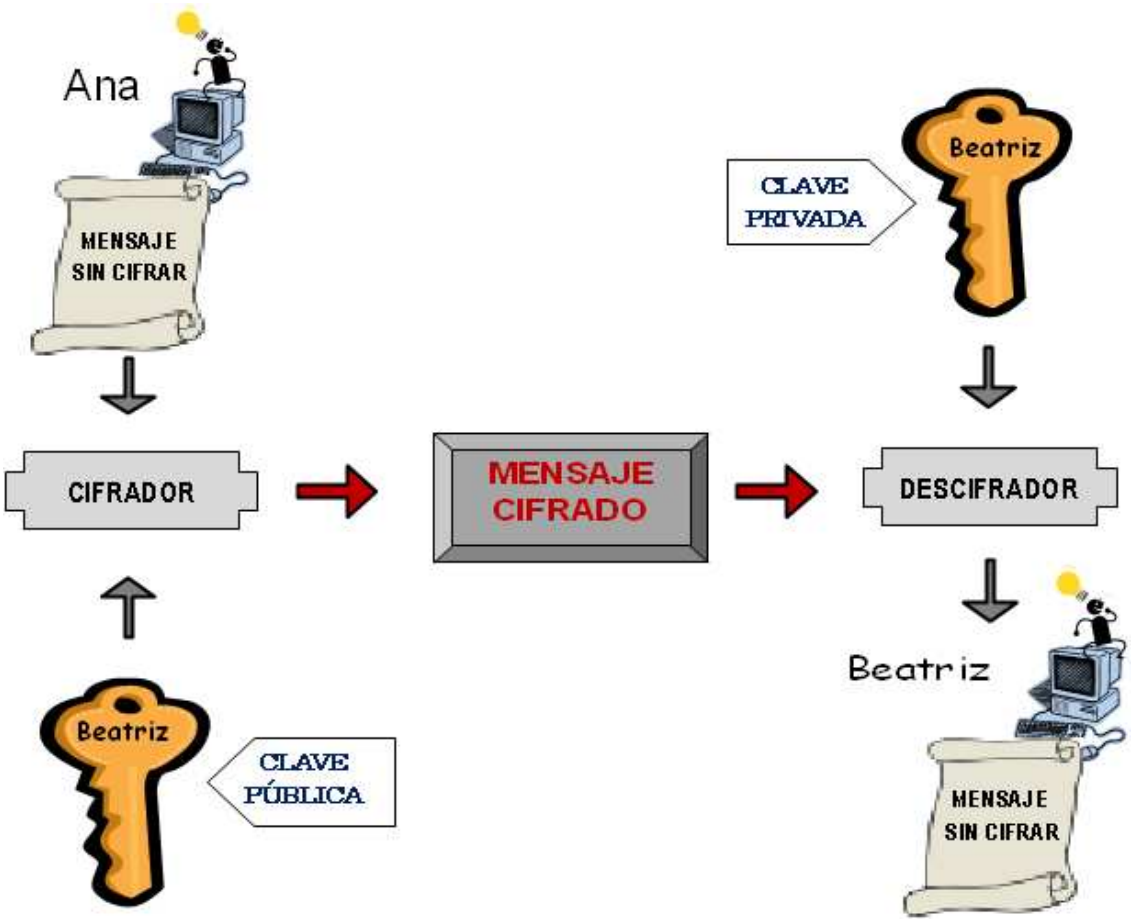

Figura 2.- Criptografía Asimétrica.

El receptor (Beatriz) de la figura 2 posee una pareja de claves relacionadas (pública y privada), y ha facilitado al emisor (Ana) su clave pública, de manera que este último puede enviarle información sensible (mensaje) cifrada con la clave pública de Beatriz, conocida por ambas partes, de tal manera que sólo Beatriz, al ser la única poseedora de su propia clave

privada, y estar ambas claves matemáticamente relacionadas, puede descifrar el criptograma y obtener la información sensible.

Algunos de los algoritmos simétricos más utilizados son AES, 3DES, CAST y BF (Blowfish), mientras que las *suites* asimétricas más extendidas son RSA, DH/DSS y ElGamal/DSA. Las funciones hash más populares son las familias SHA y MD.

3.2 Infraestructura de Clave Pública

PKI (Public Key Infrastructure) requiere un conjunto de *hardware* y *software* para su implantación, además de una serie de procesos de mantenimiento bien definidos. Surge ante la necesidad de poder gestionar todos los elementos involucrados en un proceso de certificación. Su aplicación se apoya en el estándar X.509 para Certificación Electrónica, enmarcado en los RFCs 5280 y 6818.

Dos de los elementos que integran PKI se han expuesto en el apartado anterior (la cifra clásica y la cifra moderna). En este apartado se introducen los conceptos de jerarquía de confianzas y firma digital, como elementos principales de un proceso de certificación, siendo además dos conceptos relacionados y fundamentales para poder realizar una implantación completa de la infraestructura.

Un certificado digital es un documento identificativo, que vincula a una persona o equipo con una clave pública, la cual a su vez está matemáticamente relacionada con una clave privada. La clave pública se emplea para el cifrado de información y la verificación de la firma digital,

mientras que la clave privada se utiliza para realizar las operaciones opuestas.

La firma electrónica ofrece incluso más seguridad que la manuscrita, porque no sólo es capaz de demostrar de manera irrefutable la conformidad del firmante, sino que además garantiza la integridad del documento firmado, sin poder este último ser alterado lo más mínimo. Dicho proceso de firma responde al mostrado en la siguiente figura:

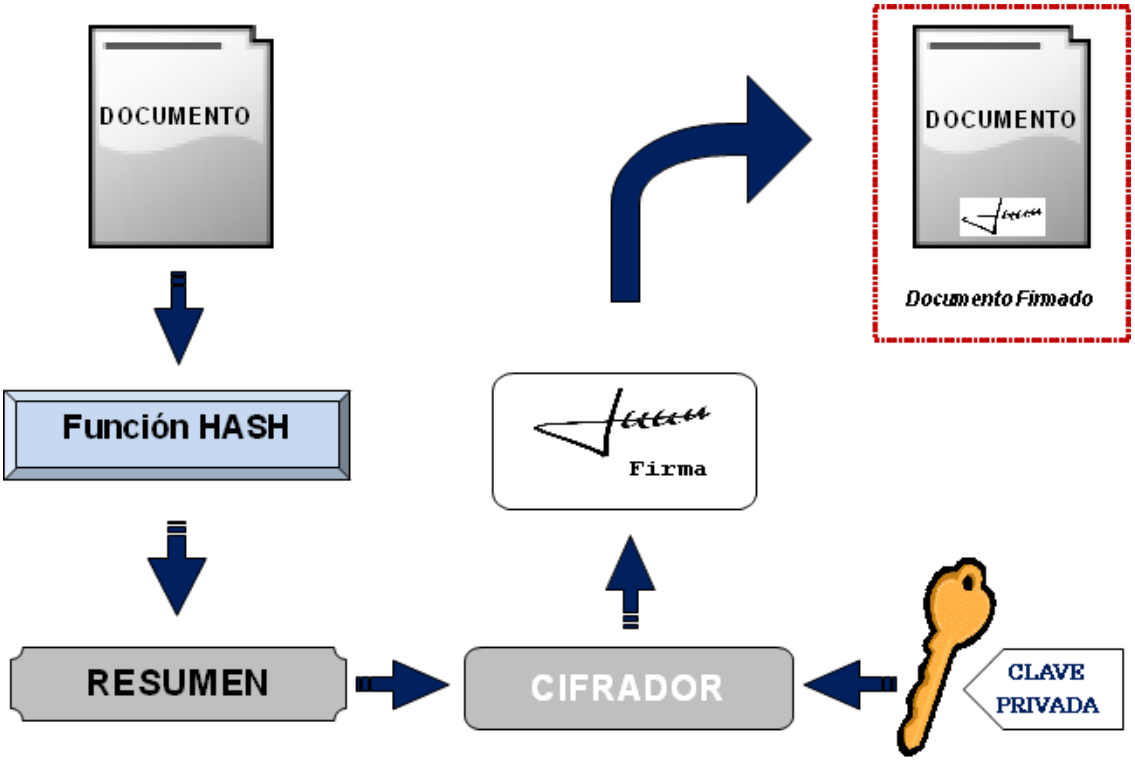

Figura 3.- Firma Electrónica.

El resumen de la figura 3 se obtiene como salida de la función hash aplicada sobre el documento, y se introduce en un cifrador asimétrico (RSA, DSA, ElGamal), que produce como resultado una cadena de bits de

longitud determinada que es conocida como la firma digital, la cual se añade al documento y se distribuye anexa al mismo.

Distribuir el documento junto con la firma posibilita que cualquiera pueda comprobar que el documento ha sido realmente firmado por el titular del certificado digital que se utilizó para realizar la firma (campo subject del certificado), y consecuentemente verificar así la identidad del firmante. Esta comprobación se realiza obteniendo la clave pública impresa en el certificado digital, que es público, y realizando una operación de verificación sobre la firma. A su vez, como el certificado digital viene firmado por una Autoridad de Certificación (CA) que confirma la veracidad de los datos contenidos en el mismo, entonces se puede asegurar con total confianza que el firmante es realmente quién dice ser, comprobando de manera similar dicha firma de la CA. Esto introduce el concepto de cadena de confianzas, en el que se sustenta la certificación electrónica, y cuyo funcionamiento se esquematiza en la siguiente figura:

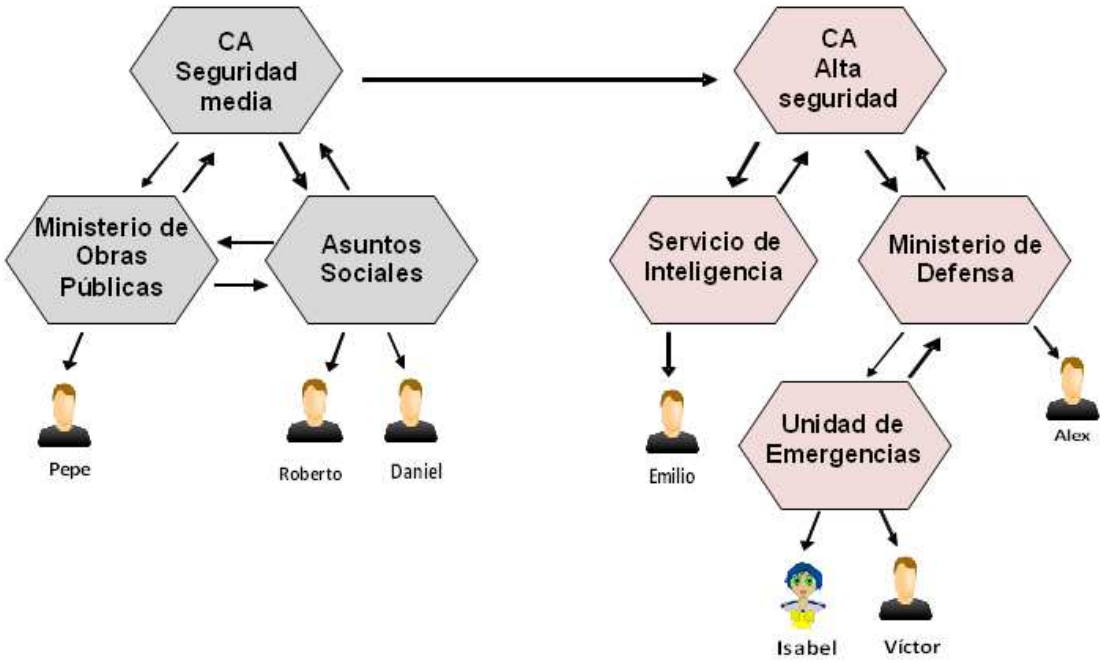

Figura 4.- Jerarquía de confianzas.

Las flechas de la figura 4 indican relaciones de confianza entre usuarios y autoridades de certificación. CA Seguridad media es una Autoridad de Certificación Raíz (como Verisign Inc. o la FNMT), que representa el nivel más alto de la jerarquía, y cuyo certificado es autofirmado (se firma así misma y todos confían en ella). El Ministerio de Obras Públicas es una autoridad subordinada, que está acreditada y certificada por la CA Raíz, luego entonces Pepe, para formar parte de esa cadena de confianzas, debe ser certificado por el Ministerio de Obras Públicas al cual pertenece, y por consiguiente su certificado debe ser firmado por dicha sub-autoridad. De esta forma, cualquier otra persona (Emilio, Daniel, Isabel) que obtenga el certificado digital de Pepe, podrá verificar que la información contenida en él, es realmente de Pepe, y no de otra persona, pues existen dos autoridades superiores que así lo ratifican y en las cuales todos ellos

confían. La autoridad subordinada avala directamente a Pepe, mientras que la CA Raíz avala a la autoridad subordinada.

3.3 Protocolo seguro de comunicaciones

Otro de los componentes de una infraestructura PKI completa es el protocolo SSL/TLS (normalmente referenciado como uno solo).

TLS (Transport Layer Security) es un protocolo de seguridad ampliamente utilizado para proteger las comunicaciones. Puede actuar sobre las capas más bajas del modelo OSI (Open System Interconnection) sin depender para ello de implementaciones de protocolo sujetas al sistema operativo, como les suele ocurrir a otros protocolos de seguridad tales como IPsec o EAP. Su versión 1.2 (la más reciente hasta la fecha) está definida en el RFC 5246, y su actualización en el RFC 6176.

SSL (Secure Sockets Layer) es el precursor de TLS, y la versión 3.0 del protocolo (la más reciente) está definida en el RFC 6101.

El algoritmo matemático de ambos protocolos es muy similar, pero TLS presenta mejoras a nivel de seguridad. Pueden hacer uso por igual de la certificación electrónica, aplicándose total o parcialmente una infraestructura PKI.

El cronograma funcional para el establecimiento de una comunicación tunelada cliente-servidor mediante SSL/TLS es el siguiente:

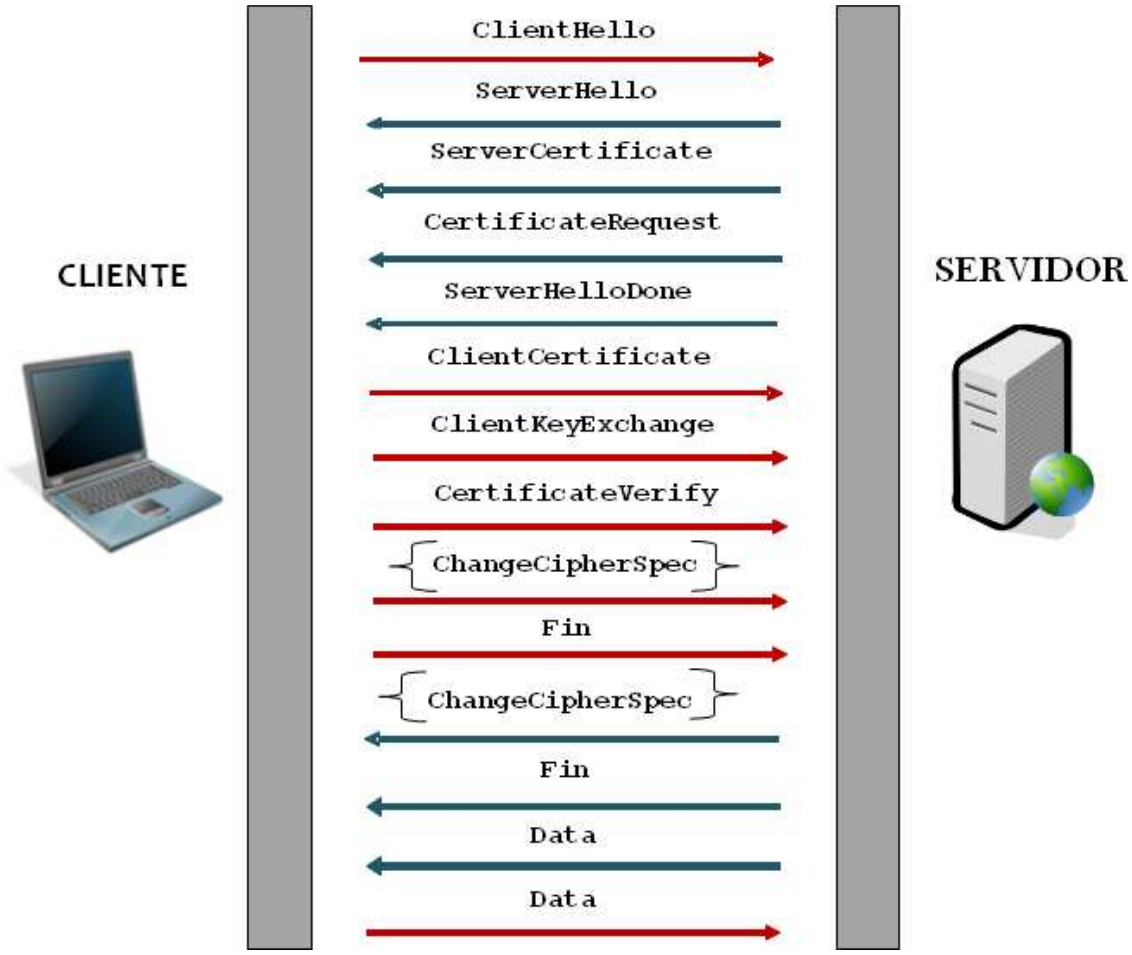

Figura 5.- Cronograma funcional SSL/TLS.

El cronograma de la figura 5 se divide en 4 fases que integran el intercambio de mensajes entre cliente y servidor, para acordar las herramientas que ambos utilizarán para el establecimiento del túnel. Todo este proceso de negociación entre ambos es conocido como el *handshake*.

En la primera fase (ClientHello), el cliente envía al servidor el protocolo y la versión del mismo que desea utilizar, además de un identificador de sesión, un valor aleatorio con alto grado de entropía, y una lista de los algoritmos criptográficos y funciones hash que soporta.

En la segunda fase (ServerHello), el servidor busca el ID de sesión en su cache de estados, y si lo encuentra inicia un *abbreviated handshake* con el cliente. En caso contrario se realiza un *full handshake*, en el cual el servidor envía al cliente un mensaje con el ID de sesión, además de la aceptación del protocolo y versión que el cliente solicitó, la *suite* criptográfica que ambos utilizarán, y un valor aleatorio del servidor. Durante esta misma fase el servidor puede enviar tres mensajes más al cliente, el ServerCertificate, el ServerKeyExchange, y el CertificateRequest.

En el mensaje ServerCertificate se envía al cliente el certificado digital del servidor, para que compruebe la firma. El mensaje ServerKeyExchange se emplea cuando la implantación de la PKI es parcial, y no se utiliza certificado de servidor. El mensaje CertificateRequest sirve para solicitar al cliente su certificado digital (implantación completa de una PKI).

La fase tercera y la cuarta engloban las respuestas del cliente al servidor y viceversa, respectivamente. En estas fases es donde se realiza el proceso de derivación de claves en paralelo, que dará como resultado el establecimiento de un canal seguro.

3.4 Tarjeta de identificación electrónica

Un componente extra de la PKI que mejora la seguridad es la *smart card* o tarjeta de identificación electrónica, que integra un chip criptográfico con sistema operativo propio. Los datos privados son almacenados en el chip, y no pueden ser vistos por nadie que no posea los credenciales necesarios. El chip es capaz de realizar cualquier operación criptográfica con total independencia del dispositivo que le da soporte. Por consiguiente, las operaciones de autenticación y firma digital pueden llevarse a cabo sin comprometer la seguridad de ningún dato secreto. Esta es su principal cualidad, proteger los datos privados, confidenciales y/o sensibles del propietario, aun en el supuesto caso de que tanto el sistema anfitrión como el propio dispositivo que le da soporte sean comprometidos.

El estándar ISO/IEC 7816, subdividido en 15 partes, describe cómo deben ser las tarjetas inteligentes y cómo deben usarse, sus características y el entorno que las rodea.

PKCS (Public-Key Cryptography Standards) es un conjunto de estándares para criptografía de clave pública, entre los que se encuentran PKCS#11 y PKCS#15, siendo estos dos sobre los que basa su implementación la tarjeta de identificación electrónica.

PC/SC es un estándar abierto para la integración de tarjetas inteligentes (ISO/IEC 7816) en entornos de computación mediante el uso de lectores.

Una representación esquemática de los elementos que componen el chip de la tarjeta puede ser la siguiente:

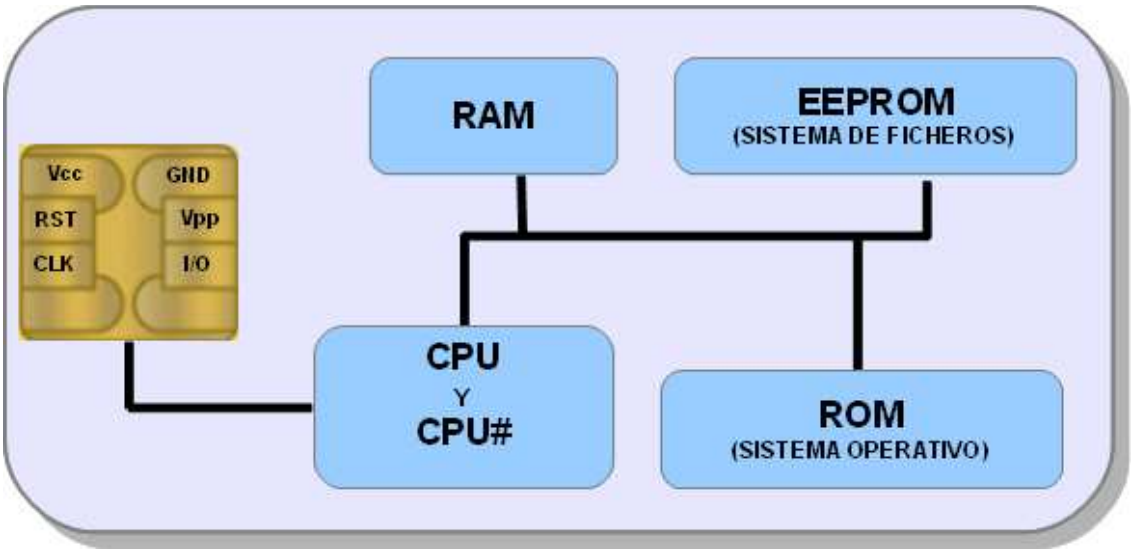

Figura 6.- Tarjeta de identificación electrónica criptográfica.

En la figura 6 se visualizan los elementos que integran el chip. La memoria ROM, al ser de sólo lectura, viene grabada de fábrica con el sistema operativo y algunos algoritmos y software de cifrado básicos, evitando así perder su contenido al desconectar el dispositivo, o ser sobrescrito. Por su parte la memoria EEPROM es más lenta que la RAM, pero permite almacenar datos que pueden perdurar tras una desconexión, hasta que el usuario decida borrarlos o sobrescribirlos, ideal para almacenar claves de cifrado, pero demasiado lenta para realizar operaciones criptográficas, para las cuales la CPU hace uso de la memoria RAM. La CPU# representa el Criptoprocesador, sobre el que se apoya la CPU para realizar las operaciones criptográficas.

3.5 Sistema de control de acceso

Una PKI se despliega en un entorno con una única finalidad, verificar las confianzas de las entidades que lo integran, antes siquiera de que estas puedan conectarse al medio (la red de datos). En este sentido se requiere un sistema de control de acceso que verifique todos los acoplamientos.

RADIUS (Remote Authentication Dial In User Service) es un estándar definido en el RFC 2865 (protocolo), en el 2924 (atributos), y en el 3576 (extensiones). Es un sistema pionero en este campo, y una de las opciones disponibles a la hora de desplegar estos controles. Su implantación más frecuente responde a la mostrada en la siguiente figura:

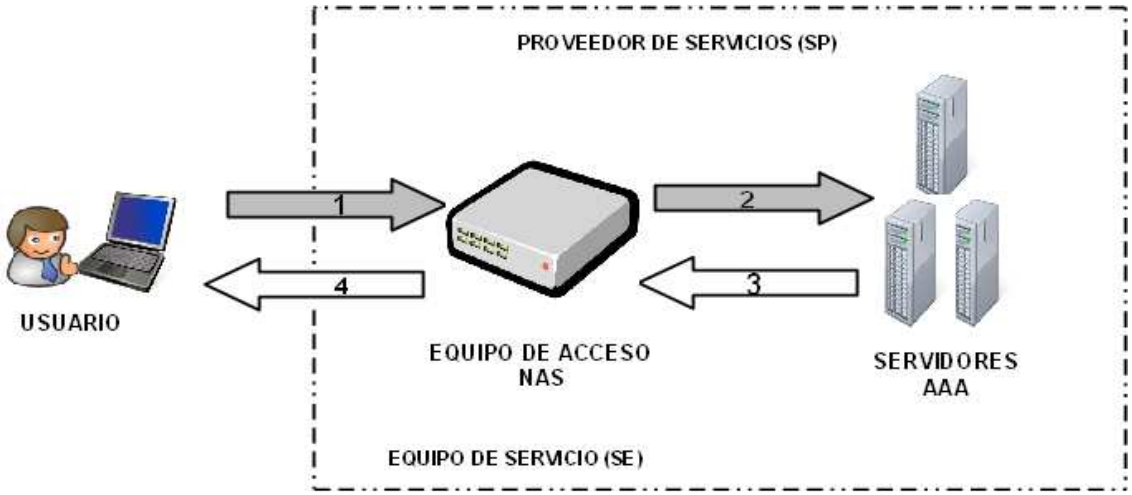

Figura 7.- Secuencia de autenticación AAA.

En la figura 7 se puede observar como un equipo de servicio de un proveedor (NAS) dirige el tráfico de un usuario hacia los servidores de autenticación. Bajo este esquema el usuario es conocido como el suplicante, el equipo de acceso como el autenticador, y los servidores AAA como servidores de autenticación.

AAA (Authentication + Autorithation + Accounting) es un estándar para el diseño de sistemas basados en la autenticación. Es una colección y definición de normas que se describen en los RFCs 2903, 2905, y 2906. RADIUS por consiguiente implementa el estándar AAA.

El protocolo RADIUS se basa en un sistema de intercambio de atributos con sus clientes, para llevar a cabo los procesos de autenticación, autorización, y arqueo. Es un protocolo de la capa de aplicación que se encapsula en paquetes UDP.

Una comunicación RADIUS sencilla responde a la mostrada en el siguiente cronograma:

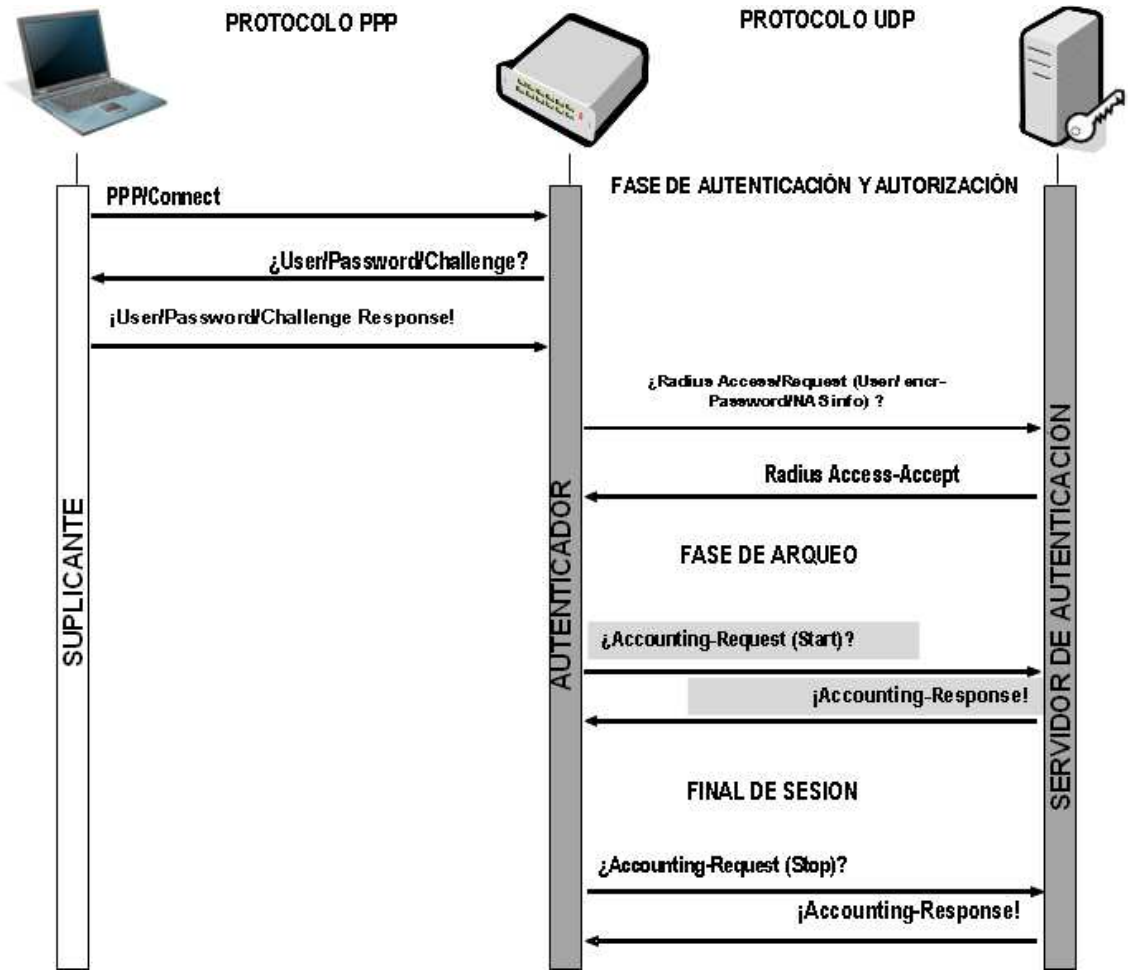

Figura 8.- Cronograma funcional de una secuencia de comunicación RADIUS.

El suplicante de la figura 8 inicia una comunicación mediante un protocolo punto a punto con el autenticador para conseguir el acceso a la red. Los credenciales solicitados al cliente (suplicante) son enviados por el autenticador al servidor de autenticación, quién los verifica contra su base de datos y permite al autenticador conceder el acceso (mensaje Access-Accept). Tras ello se realiza la fase de arqueo, donde se monitoriza la conexión del cliente hasta que esta finaliza.

3.6 Extensión de autenticación

El problema de basar los procesos de autenticación en protocolos de la capa de aplicación (como RADIUS) es que se permite una comunicación abierta sobre las capas más bajas del modelo OSI. Debido a esto se incorporó a estos procesos el estándar 802.1x, definido en el RFC 3580, y cuyo representante más popular es el protocolo EAP (Extensible Authentication Protocol), que actúa sobre la capa 2 del modelo OSI. Este protocolo permite controlar el acoplamiento de cualquier sistema al medio, y para ello puede transportar desde mecanismos de autenticación simples hasta los más complejos y robustos, basados en certificados PKI.

El estándar 802.1x implementa el control de acceso a la infraestructura de red mediante el PAE (Port Access Entity), que divide un puerto físico en dos puertos virtuales (controlado e incontrolado), permitiendo exclusivamente tráfico dirigido al servidor de autenticación por el puerto incontrolado (EAP), hasta que la identidad del suplicante es verificada por el servidor, y entonces el puerto controlado que da acceso a la red es abierto para su uso por parte del cliente. Tal y como se muestra en la figura 10.

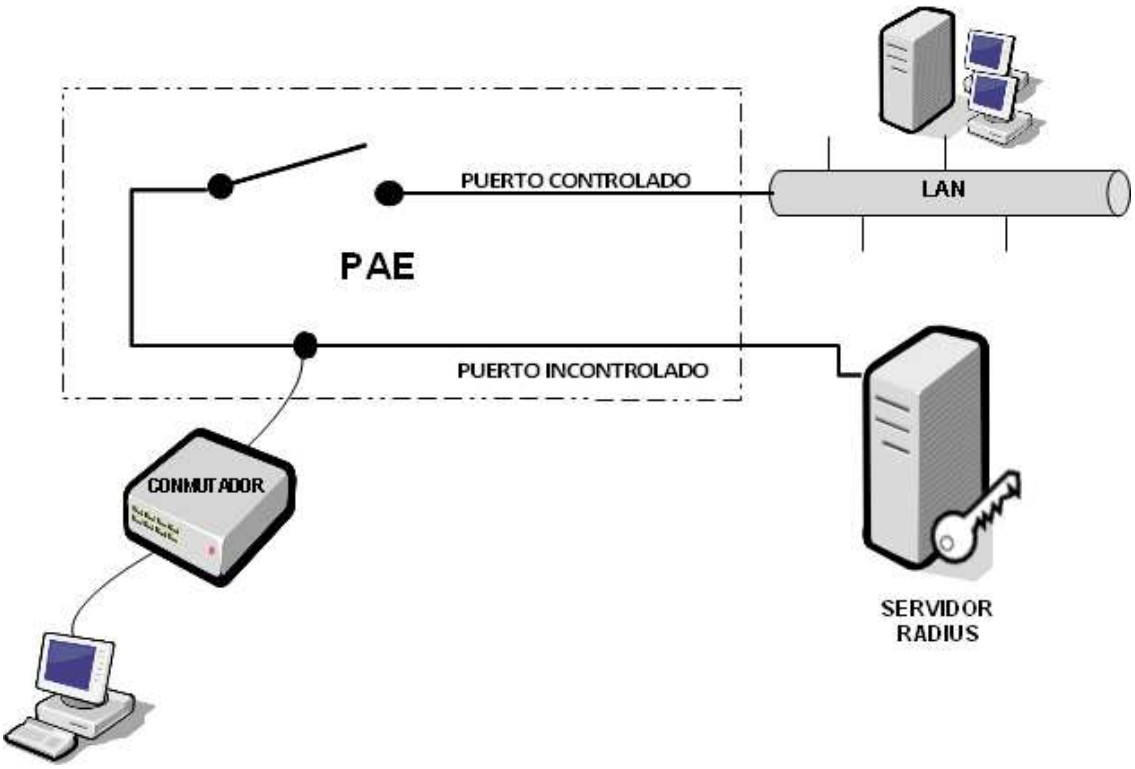

Figura 9.- Estado de puertos 802.1X.

Al igual que RADIUS el protocolo EAP basa su operación en el intercambio de atributos, que sirven para definir, establecer, y regular, el acceso del cliente a la red.

Entre los mecanismos de autenticación que facilita EAP se encuentran métodos simples como PAP, CHAP, o MD5, y los más seguros como EAP-TLS, EAP-TTLS, y EAP-PEAP, que hacen uso del protocolo TLS (expuesto en el apartado 3.3) para implementar túneles cifrados mediante certificados, fortaleciendo así realmente las comunicaciones.

El siguiente cronograma funcional muestra como se realiza una autenticación mediante EAP:

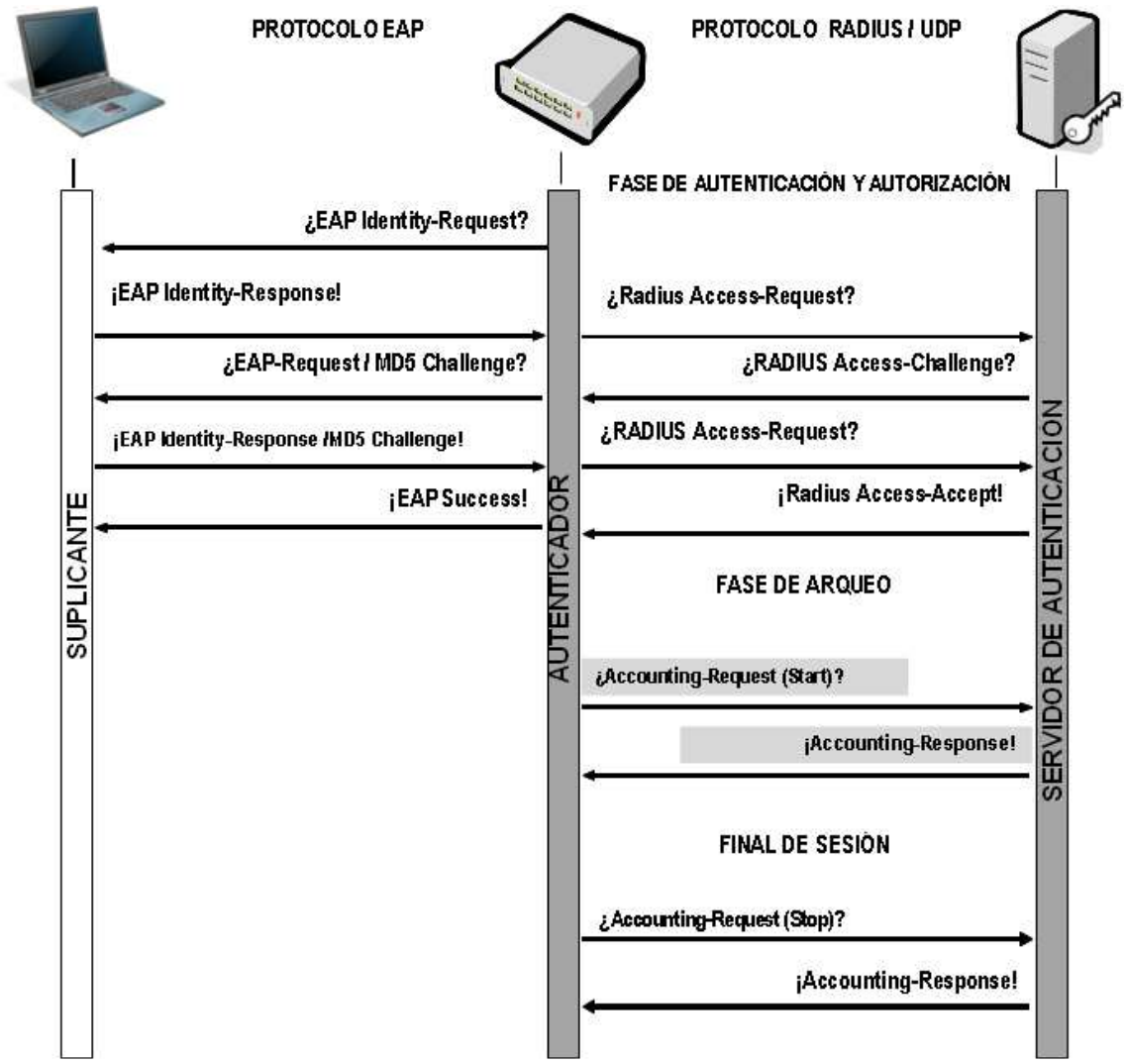

Figura 10.- Cronograma funcional de una secuencia EAP sobre RADIUS.

El mecanismo de autenticación utilizado en la figura 10 es EAP-MD5 (actualmente sólo recomendable en entornos muy controlados), y el proceso es muy similar al mostrado en el apartado anterior. El protocolo EAP es encapsulado en tramas ethernet en la comunicación entre el suplicante y el autenticador, lo cual es conocido como el protocolo EAPoL. El autenticador no entiende el protocolo EAP y simplemente se limita a encapsularlo en paquetes RADIUS, y posteriormente enviárselos al servidor de autenticación.

3.7 Red de cobertura inalámbrica

La infraestructura expuesta hasta el momento responde perfectamente a una arquitectura basada en acceso cableado, pero en la actualidad, es habitual ofrecer más opciones para conectarse a una infraestructura de red. En este sentido se puede integrar una arquitectura WiFi en la infraestructura, basando su seguridad en la normativa 802.11i, recogida en el RFC 4017, que abarca el estándar 802.1x expuesto en el apartado anterior.

Las fases en las que se desglosa un proceso de conexión regulado por esta normativa se muestran en la siguiente figura:

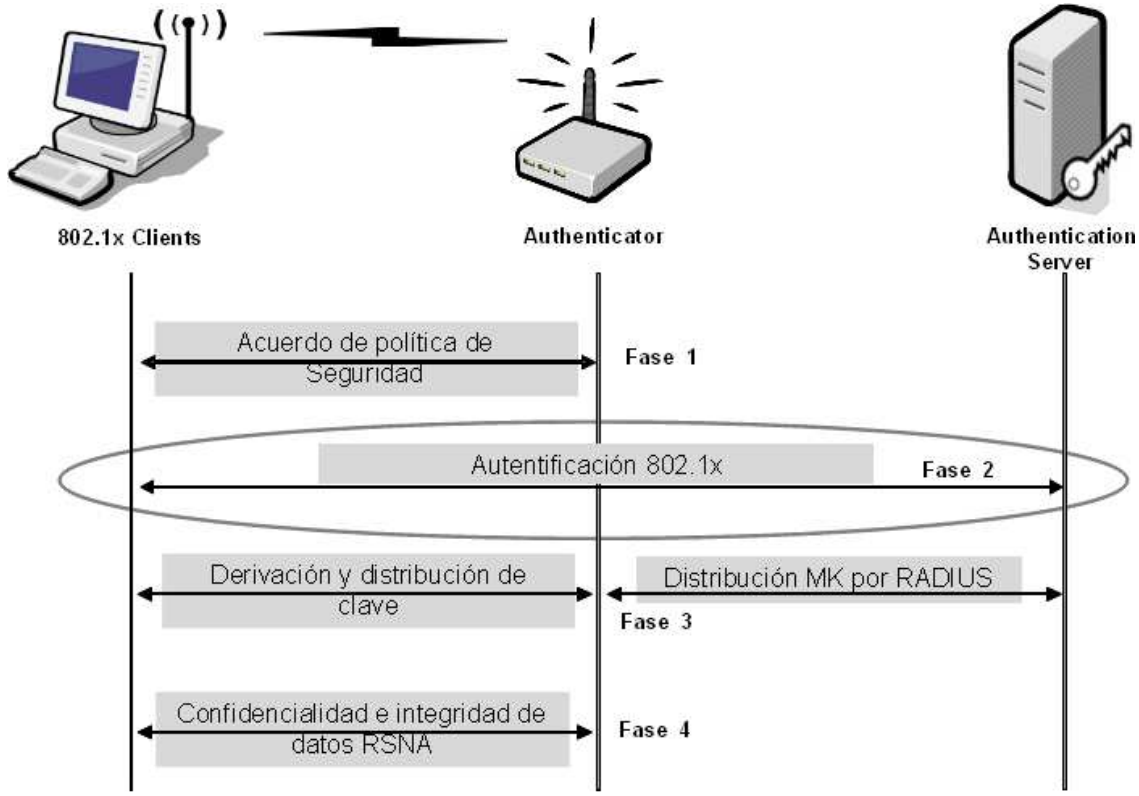

Figura 11.- Secuencia de fases 802.11i.

La segunda fase de la figura 11 responde a lo ejemplarizado en el apartado anterior sobre EAP (802.1x), mientras que las fases uno, tres, y cuatro, son propias de la normativa de seguridad inalámbrica 802.11i, en la cual se detalla un proceso de derivación y distribución de claves que da lugar a distintos modelos de seguridad.

En lo referente a los modelos de seguridad existen dos tipos de implantación corporativa, la arquitectura RSN y la TSN, ambas recogidas en la normativa. RSN es más conocida como WPA2-Enterprise, y utiliza AES-CCMP (WPA2) para el cifrado de datos, mientras que TSN implementa un modelo de compatibilidad con productos antiguos actualizados mediante firmware, aceptando tanto WPA2 como WPA (TKIP-RC4).

Una conexión Wi-Fi incluyendo todos estos elementos responde a la mostrada en el siguiente cronograma:

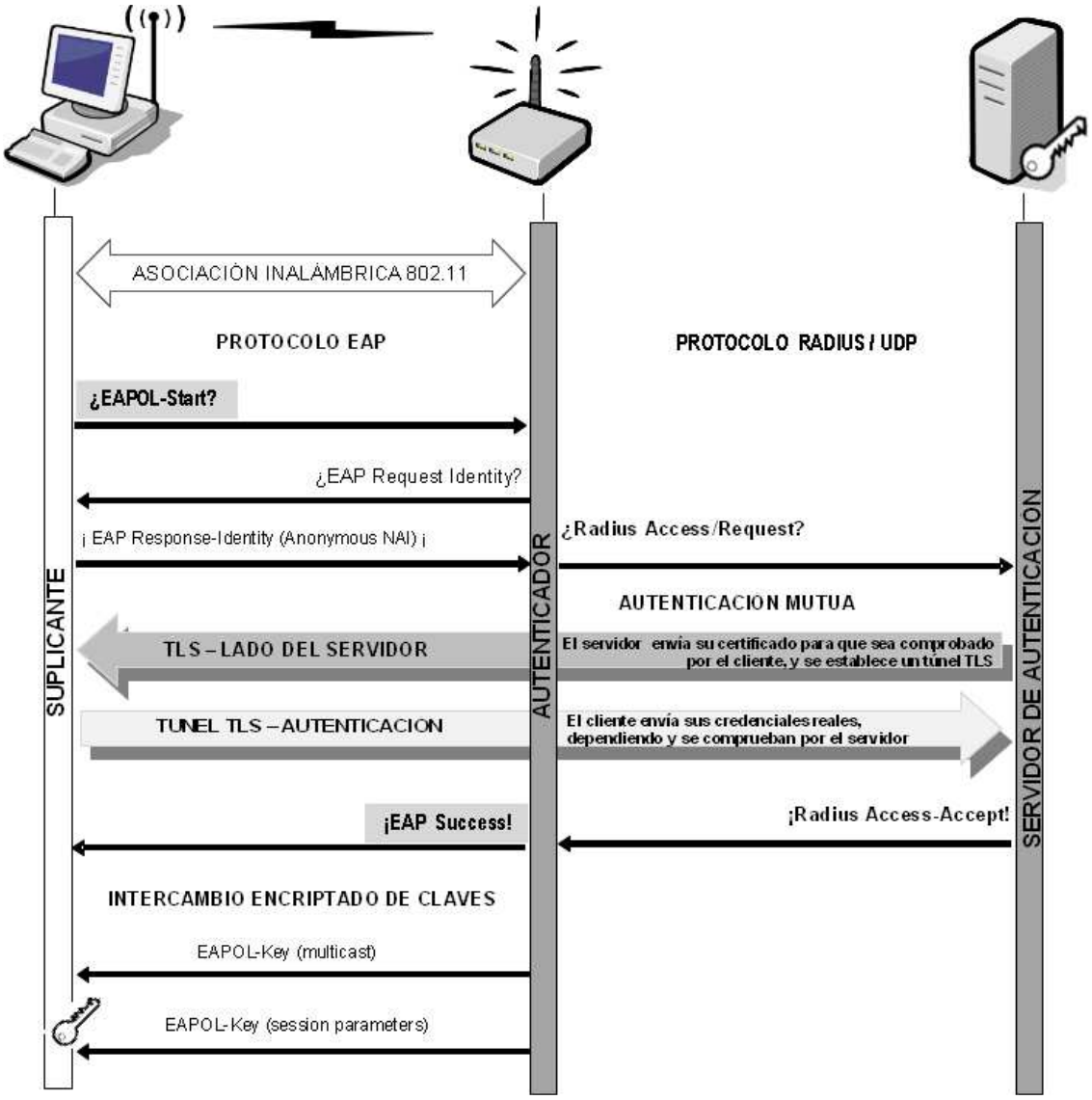

Figura 12.- Cronograma funcional de una comunicación 802.11i.

El método EAP utilizado en la figura 12 es EAP-TLS. El túnel cifrado establece un proceso de autenticación mutua mediante certificados PKI, y en su interior tiene lugar el intercambio de material que se emplea en el proceso de derivación de claves posterior, para que ambos, suplicante y autenticador, utilicen dichas claves para proteger la sesión de usuario.

3.8 Red privada virtual

Como complemento y extensión de la infraestructura descrita se pueden implementar enlaces VPN para el fortalecimiento de la misma. Estos enlaces pueden ser tanto internos como externos, dando lugar a *intranets* blindadas y a accesos remotos seguros, respectivamente.

Estos túneles cifrados pueden implementarse sobre la capa de red o sobre la capa de transporte (modelo OSI). En esencia el resultado es el mismo, pero su utilidad, el rendimiento, y la implementación, son distintos.

Un esbozo de protocolos de seguridad en referencia a los niveles del modelo OSI puede visualizarse en la siguiente figura:

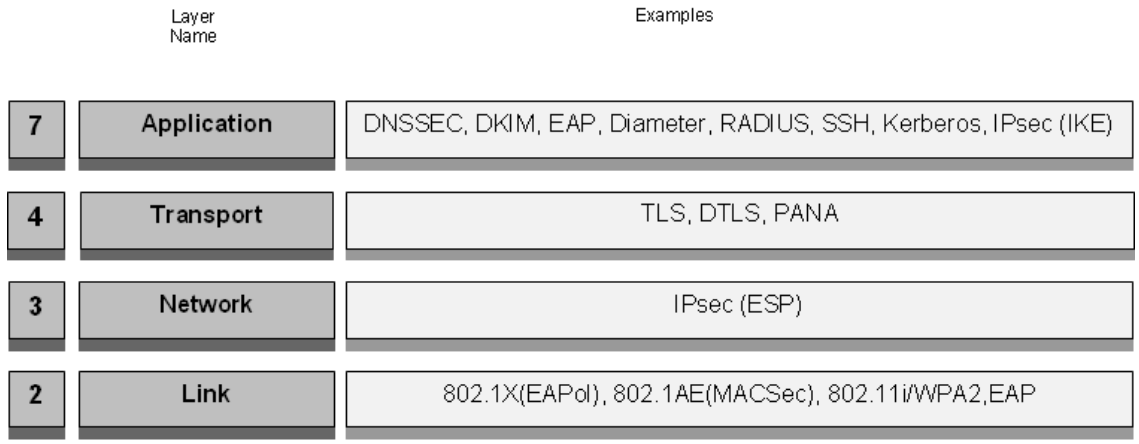

Figura 13.- Jerarquía de niveles y protocolos de seguridad.

Como se aprecia en la figura 13, IPsec y TLS (ambos protocolos mencionados en apartados anteriores) se aplican sobre capas distintas. Esto permite mantener enlaces en distintos niveles. Por lo general, los enlaces internos se implementan sobre la capa 3, mientras que los externos

se mantienen en la capa 4, debido a la facilidad que permite TLS para hacer y deshacer enlaces.

La lógica de diseño es siempre la misma. El paquete de la capa sobre la que se implementa el túnel se encapsula de nuevo sobre la misma capa, dando lugar a un *payload* que contiene otro paquete de la misma capa, con la diferencia de que ese *payload* va cifrado, permitiendo así la protección del canal de datos e implementando la VPN.

4. DESCRIPCIÓN DEL PROYECTO

4. Descripción del proyecto

La implantación de este sistema de autenticación global es adecuada en entornos muy diversos, desde un proveedor de servicios de Internet que quiere gestionar el acceso de sus clientes, hasta un organismo gubernamental que requiere de un riguroso proceso de autorización en el acceso a información confidencial.

El diseño e implementación descritos en el presente texto están enfocados a su implantación en una entidad corporativa que desee controlar la conexión de sus empleados a su red de datos, y como consecuencia de ello, a sus servicios internos.

En esencia el sistema global está constituido por cinco elementos, claramente diferenciados entre sí por las funciones que realizan. Son los siguientes:

- Servicio de autenticación (Radius).
- Servicio de acceso cableado (NAS).
- Servicio de acceso inalámbrico (AP).
- Servicio de acceso remoto (VPN).
- Servicio de certificación electrónica (CA).

Todos estos elementos se van a implantar mediante *appliances* (dispositivos *hardware/software* con *firmware*) y plataformas BSD (UNIX). Pero dado que las aplicaciones seleccionadas para ofrecer los servicios son multiplataforma, la elección de la plataforma en cuestión es poco relevante,

y por lo tanto en este texto se realiza una abstracción completa de implementaciones y referencias técnicas de sistema operativo.

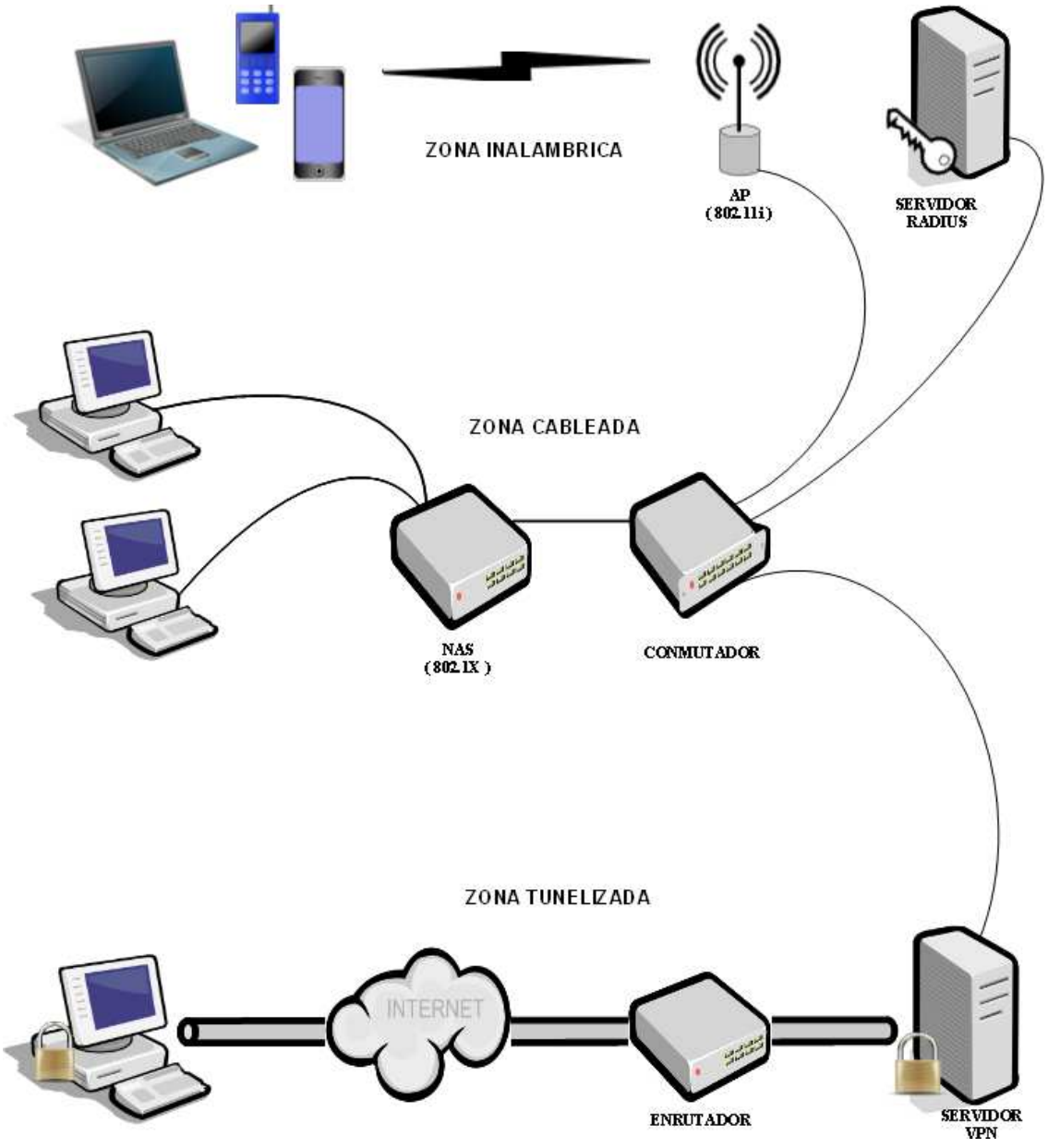

Figura 14.- Infraestructura básica para el control de acceso global.

Las características, interacción, y funcionalidad de los elementos del sistema, se detallan a continuación:

- Plataforma FreeBSD/OpenBSD portable (en *token* USB). Debidamente replicada y almacenada en lugar seguro, pues la información contenida, en especial la clave privada de la CA, es extremadamente sensible y debe ser confidencial. Esta plataforma actuará como servicio de certificación electrónica (CA), y por consiguiente debe contener una instalación básica del sistema operativo, además de las *suites* OpenSSL y OpenSC, para el manejo de certificados digitales y dispositivos criptográficos (Smart Card), respectivamente.

- Plataformas OpenBSD, con el paquete base y sin *software* gráfico. Una de las cuales contendrá una instalación completa del aplicativo FreeRadius, que implementa el estándar AAA, para la gestión de los procesos de autenticación, autorización, y arqueo. La otra dispondrá de una *suite* OpenVPN, para implementar el control de acceso remoto (desde una red pública). Ambas plataformas compartirán parte del *bundle* completo de certificación electrónica (certificado de autoridad y CRL), a excepción del certificado de servidor, que debe ser distinto debido al campo CommonName.

- Servidor de acceso a redes (NAS) con soporte 802.1X, que implementa protocolos de extensión de autenticación (en este caso se hará uso de EAP). Este *appliance* se encargará de controlar el enlace (*layer* 2) cableado a la red mediante el mecanismo PAE (Port Access Entity). Estará en constante comunicación con el servidor de autenticación, y debido a ello no necesita disponer de ningún tipo de certificación electrónica.

- Punto de acceso inalámbrico (AP) que implementa la normativa 802.11i (incluye 802.1x), para seguridad Wi-Fi (Wireless Fidelity). La finalidad del AP es facilitar la conexión a la infraestructura de la empresa a aquellos empleados que bien por limitaciones de sus dispositivos (terminal móvil, tableta), o bien por su ubicación física, no puedan hacer uso de otros medios de conexión. Este *appliance* al igual que el equipo NAS, no requiere *bundle* de certificación electrónica.

Además de los citados elementos que forman la infraestructura, para poder poner a prueba el sistema es necesario tener al menos un cliente que haga uso de los servicios mencionados. A tal efecto se dispondrá de una plataforma FreeBSD con un entorno gráfico completo y las *suites* OpenVPN (que integra cliente y servidor) y OpenSC. El paquete de software wpa_supplicant (suplicante), que implementa soporte de cliente para los estándares 802.11i y 802.1x, viene integrado en el sistema base.

Aunque para esta implantación no se hará uso de un entorno gráfico, el cliente puede optar por herramientas GUI (Graphical User Interface) como wifimgr y openvpn-admin, para la gestión y configuración de las conexiones Wi-Fi y VPN respectivamente.

Un diseño simple de la infraestructura comentada responde al mostrado en la figura 14, donde el AP implementa seguridad WPA2-Enterprise basada en arquitectura RSN (Robust Security Network), autenticando la identidad de los clientes contra el servidor RADIUS, al igual que hace el equipo NAS, mientras que el servidor VPN autentifica todas las conexiones remotas por sí mismo. Para realizar los procesos de identificación y autenticación,

ambos sistemas (VPN y RADIUS) se apoyan en la PKI desplegada por la CA, y consecuentemente en el estándar de certificación electrónica (x.509).

En una instalación productiva el proceso de mantenimiento y emisión de tarjetas de identificación electrónica para los empleados (con sus correspondientes certificados y claves), podría ser a grandes rasgos el siguiente:

1. Un empleado, bien por pérdida, robo, o nueva incorporación, solicita al departamento de Infraestructuras acceso a los recursos de la empresa, para poder realizar su trabajo. Inicialmente se le concederá, tras la debida gestión de trámites, acceso cableado e inalámbrico. Si el empleado dispusiera de la modalidad de tele-trabajo, entonces se le concedería además acceso remoto.

2. Una vez aprobada la solicitud, el responsable de sistemas hará uso de la plataforma que implementa la CA (un entorno limpio, seguro, y portable, pudiendo incluso contener una partición raíz cifrada), para en cualquier ubicación, bien sea en un lugar específico de la empresa o en el propio puesto de trabajo del empleado, crear las nuevas claves, generar los certificados, e integrarlo todo en el dispositivo criptográfico, pudiendo este último, ser incluso grabado externamente con datos identificativos del empleado, como un primer y superfluo nivel de identificación.

3. Seguidamente un empleado de nueva incorporación podría configurar las conexiones de su equipo, haciendo uso de los manuales de la

empresa, o lo que es más lógico, disponer de un equipo previamente configurado a tal efecto.

4. Finalmente el empleado podría hacer uso de las herramientas gráficas mencionadas con anterioridad, o de cualquier otra que se considere oportuno, para activar o desactivar las conexiones a su voluntad.

En base a lo expuesto, un empleado hará uso de su tarjeta de identificación electrónica corporativa para acceder a la infraestructura de red de la compañía, y el acceso le será concedido en función del estado de sus credenciales (contenidos en la tarjeta). Si están caducados, revocados, corrompidos, o aparentemente han sido falsificados, el acceso le será denegado, en cualquier otro caso se le concederá el acceso y se le asignaran los parámetros pertinentes para regular su sesión de usuario en el sistema.

5. REALIZACIÓN DEL PROYECTO

5. Realización del proyecto

En este capítulo se detalla el proceso de instalación, configuración, despliegue y puesta en funcionamiento, del sistema de control de acceso global a redes de datos mediante tarjeta de identificación electrónica descrito en el capítulo anterior.

5.1 Requisitos y especificaciones

Todo el *software* empleado para la realización de este proyecto es de libre distribución, y ha sido seleccionado por ofrecer una gran calidad, pro-activa seguridad, buena integración, compatibilidad, portabilidad, y una excelente comunidad de soporte.

Los requisitos *software* y las especificaciones *hardware* de todos los productos utilizados para realizar esta implementación se detallan a continuación:

- Servidor de autenticación FreeRADIUS v2.2.x (Servidor AAA), desplegado sobre una plataforma OpenBSD i386 v5.x (OS), la cual integra soporte criptográfico (OpenSSL) en su *software* base.

- Servidor de certificación electrónica (CA), implementado mediante la *suite* criptográfica OpenSSL v1.0.1x, y desplegado en una plataforma FreeBSD i386 v9.x (OS).

- Punto de acceso inalámbrico (AP) compatible con el estándar IEEE 802.11i, y actuando como un autenticador Wi-Fi contra el servidor

AAA. Implementado mediante un *router* Wi-Fi Comtrend Gigabit, que posee las siguientes características:

- Command line management
- Integrated 802.11n/b/g AP
- WPA and 802.1x/ WPS 2.0/ RADIUS client
- DHCP Server/Client
- Static route/ RIP/ RIP v2 routing function
- DNS Proxy/Relay

o Servidor de acceso cableado (NAS) compatible con el estándar IEEE 802.1X, y actuando como un autenticador ethernet contra el servidor AAA. Implementado mediante un *switch* Cisco Catalyst 2950, que posee las siguientes características:

- IEEE 802.1x support
- IEEE 802.3x full duplex on 10BASE-T and 100BASE-TX ports
- IEEE 802.1D Spanning-Tree Protocol
- IEEE 802.1Q VLAN
- Ports: 12 x RJ45
- Speed: 10/100Mbps

o Servidor de acceso remoto OpenVPN v2.3.x (Servidor VPN), desplegado sobre una plataforma OpenBSD i386 v5.x (OS), la cual integra soporte criptográfico (OpenSSL) en su software base. El servidor está directamente conectado a la red pública (WAN) o detrás de un dispositivo encaminador conectado a dicha red.

- El equipo de empleado consta de una plataforma FreeBSD i386 v9.x (OS), con un suplicante WPA_Supplicant v2.x (Wi-Fi/Wired), un cliente OpenVPN v2.3.x, una *suite* OpenSC v0.13.x (PKCS#11/15), y la *suite* criptográfica OpenSSL v1.0.1x con el módulo engine_pkcs11, para permitir a las aplicaciones operar con la tarjeta de identificación electrónica. Además de estos elementos, para el soporte del lector USB se debe incluir el controlador PC/SC-CCID (pcsclite).

  El lector de tarjetas inteligentes utilizado en esta implementación es un OMNIKEY CardMan 3021 USB, con las siguientes características:

  - USB 2.0 CCID (compatible con USB 1.1)
  - Soporta T=0, T=1
  - Compatible con ISO 7816 and EMV 2000
  - PC/SC driver (release 2.0)
  - CT-API (sobre PC/SC)
  - Synchronous-API (sobre PC/SC)
  - OCF (sobre PC/SC)

  La tarjeta criptográfica es una Feitian PKI (FTCOS/PK-01C), muy bien soportada por la comunidad de OpenSC. Posee un chip incrustado que hace uso de un driver *entersafe* (pcsclite compatible), el cual está disponible bajo licencia de software libre. Sus características técnicas son las siguientes:

  - Soporta T=0, T=1 o USB.
  - Código PIN de seguridad.

- Cumple ISO/IEC 7816 y PKCS#15.
- Espacio de usuario de 64KB.
- Integra RSA, DES, 3DES, SHA1.
- Claves RSA de hasta 2048 bits.
- Soporta PKCS#11 y MS-CAPI.
- Estándar X.509 v3.
- Generador de números aleatorios.
- Compatible con OpenSC.

En lo referente a la tarjeta de identificación electrónica criptográfica (Smart Card) y al lector de tarjetas USB, todos aquellos dispositivos soportados por la suite OpenSC y el controlador PC/SC-CCID deben ser totalmente funcionales y compatibles con esta implementación, y por consiguiente podrían perfectamente reemplazar a los utilizados.

5.2 Creación de certificados digitales

Para desplegar la infraestructura comentada según las premisas expuestas, se requieren tres categorías de certificación:

- Certificación de CA.
- Certificación de Servidor(es).
- Certificación de Cliente.

El procedimiento habitual de certificación consiste en la generación de una CSR (Certificate Sign Request) por parte del cliente. Dicha solicitud es enviada a la CA para su correspondiente verificación y firma, generándose

así el certificado digital que esta última devuelve al cliente (solicitante). Pero para esta PKI particular es más lógico simplificar el proceso, manteniendo al empleado fuera del mismo, y siendo la CA (integrada en el dispositivo portable) quién se encargue total y exclusivamente de todo el proceso de emisión y mantenimiento.

Junto con el *software* criptográfico instalado (OpenVPN, FreeRadius, OpenSSL), se incluye una serie de *scripts* que simplifican todo el proceso de certificación y mantenimiento, como es el caso de easy-rsa. Pero en este texto se emplea directamente la línea de comandos de openssl.

Antes de generar los certificados es necesario modificar ligeramente el archivo de configuración por defecto */usr/local/openssl/openssl.cnf*, en base a los requisitos y preferencias:

1. En la sección *[CA_default]* se cambian los siguientes valores:

```
dir = ./                        # where everything is kept
database = $dir/index
new_certs_dir = $dir/store      # place for new certs.
certificate = $dir/cacrt.pem    # The CA certificate
# crlnumber = $dir/crlnumber
default_days = 730              # how long to certify for
default_bits = 2048
private_key = $dir/cakey.pem
RANDFILE = $dir/.rand
```

2. En la sección *[v3_ca]* se modifican las siguientes extensiones:

```
basicConstraints = critical, CA:true, pathlen:0
keyUsage = cRLSign, keyCertSign
subjectAltName = email:copy
# authorityKeyIdentifier = keyid:always,issuer
```

3. Se crea la sección *[server]* con atributos de autenticación*:*

```
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyAgreement
extendedKeyUsage = serverAuth
nsCertType = server
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid
```

4. Se crea la sección *[client]* que es similar a la del servidor*:*

```
basicConstraints = CA:FALSE
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid
```

Ahora es necesario inicializar la base de datos y el número de serie para los nuevos certificados, además de crear su directorio de almacenamiento. Todo tal cual se ha especificado en el archivo de configuración de openssl. Para ello se ejecuta lo siguiente en la *shell* de usuario:

```
$ echo 01 > serial && touch index && mkdir store
```

A continuación se va a generar la siguiente lista de ficheros relevantes:

- cacrt.pem > certificado de CA.
- cakey.pem > clave privada de CA.
- radcrt.pem > certificado de servidor RADIUS.
- radkey.pem > clave privada de servidor RADIUS.
- tuncrt.pem > certificado de servidor VPN.
- tunkey.pem > clave privada de servidor VPN.
- clncrt.pem > certificado de cliente.
- clnkey.pem > clave privada de cliente.
- crl.pem > lista de revocación de certificados.

Para generar el certificado *self-signed* de la CA se le da la siguiente orden al intérprete de comandos:

```
$ openssl req -x509 -set_serial 0x0 \
-newkey rsa:4096 -sha512 -days 1825 \
-passout pass:1234 \
-subj "/C=ES/O=SisCE/OU=SecRootCA/emailAddress=sec@sisce.esc" \
-keyout cakey.pem -out cacrt.pem
```

Para generar los credenciales correspondientes para el servidor VPN la invocación completa a la *suite* criptográfica es la siguiente:

```
$ openssl req -new -passout pass:1235 \
-subj "/C=ES/O=SisCE/OU=SecVPN/CN=tun.sisce.esc" \
-keyout tunkey.pem -out tuncsr.pem && openssl ca \
```

```
-batch -policy policy_anything -passin pass:1234 \
-extensions server -out tuncrt.pem -infiles tuncsr.pem
```

Para generar los credenciales del servidor RADIUS la instrucción es:

```
$ openssl req -new -passout pass:1235 \
-subj "/C=ES/O=SisCE/OU=SecAUTH/CN=rad.sisce.esc" \
-keyout radkey.pem -out radcsr.pem && openssl ca \
-batch -policy policy_anything -passin pass:1234 \
-extensions server -out radcrt.pem -infiles radcsr.pem
```

Ahora se debe eliminar el *passphrase* de las claves privadas de ambos servidores (requerido durante la generación), para que puedan realizar sus funciones de manera desatendida, y sin necesidad de almacenar claves en ficheros de configuración:

```
$ openssl rsa -passin pass:1235 -in tunkey.pem -out tunkey.pem
$ openssl rsa -passin pass:1235 -in radkey.pem -out radkey.pem
```

Para generar los credenciales de un cliente:

```
$ openssl req -new -passout pass:1235 \
-subj "/C=ES/O=SisCE/OU=Preventa/CN=JIMENEZ PINTO, IGNACIO" \
-keyout clnkey.pem -out clncsr.pem && openssl ca \
-batch -policy policy_anything -passin pass:1234 \
-extensions client -out clncrt.pem -infiles clncsr.pem
```

El *passphrase* para la clave privada del cliente también debe ser suprimido, porque cuando se almacene la clave en la tarjeta electrónica el propio PIN de la misma se encargará de protegerla:

```
$ openssl rsa -passin pass:1235 -in clnkey.pem -out clnkey.pem
```

A continuación y por comodidad, se genera un *bundle* con formato PKCS12, almacenando tanto los credenciales del cliente (clave pública/privada) como el certificado de la CA. De esta manera se puede importar todo el conjunto de una sola vez en la tarjeta. Para la generación del *bundle* se ejecuta lo siguiente:

```
$ openssl pkcs12 -export -passout pass:1235 \
-out clnpkg.p12 -inkey clnkey.pem \
-in clncrt.pem -certfile cacrt.pem
```

Ahora en primer lugar se limpia e inicializa la tarjeta criptográfica, y tras ello se almacenan los credenciales previamente generados. Después de activar el *daemon* pcscd (con privilegios de *root*), conectar el lector, e insertar la tarjeta, se ejecutan las siguientes instrucciones:

```
$ pkcs15-init --erase-card && pkcs15-init \
--create-pkcs15 --profile pkcs15+onepin \
--pin 1234 --puk 123456
$ pkcs15-init --store-private-key clnpkg.p12 \
--format pkcs12 --auth-id 01 --pin 1234 \
--passphrase 1235
```

Una vez finalizado todo el proceso se deberían eliminar del directorio raíz todos los archivos con extensión “.csr” (Certificate Sign Request), pues son totalmente innecesarios. Los archivos “.pem” de cliente también pueden ser eliminados, pues se dispone del *bundle* pkcs12, que puede ser

renombrado al ID correspondiente asignado por la CA en el directorio *./store,* y seguidamente almacenado en un nuevo directorio de certificados de cliente, para una futura posible reutilización.

La lista de revocación de certificados (CRL) necesaria para que los servidores puedan controlar la autorización en el acceso puede ser generada con el siguiente comando:

```
$ openssl ca -gencrl -passin pass:1234 -out crl.pem
```

Inicialmente la lista estará vacía hasta que no se revoque algún certificado y se genere nuevamente la misma. Para la revocación de certificados se debe ejecutar:

```
$ openssl ca -passin pass:1234 -revoke ./store/XX.pem
```

El valor XX representa el número del certificado (de entre los disponibles en el almacén) que se quiere invalidar. La ejecución de la siguiente instrucción de actualización sobre la base de datos de certificados (fichero index) también es adecuada para revocar todos aquellos certificados que hayan expirado (superado su fecha de vencimiento), y puede ser ejecutada con regularidad, aunque igualmente el servidor se encarga de verificar las fechas:

```
$ openssl ca -passin pass:1234 -updatedb
```

Como se puede observar, una CA requiere un mantenimiento periódico, y por tanto una atención constante. Pero todos estos procesos pueden ser igualmente automatizados, produciendo como resultado un sistema simple,

sencillo, y limpio, a la vez que seguro y funcional, lo cual convierte esta implementación en una buena elección para infraestructuras de tamaño medio. De cara a afrontar el mantenimiento de implementaciones de gran volumen existen herramientas como un OCSP Responder o/y OpenCA, que intentan mejorar la gestión de todos los procesos implicados.

5.3 Despliegue de acceso cableado

En primer lugar se debe eliminar el contenido del directorio */etc/raddb/certs* del servidor. De los datos generados en el capítulo previo, el certificado de CA y los credenciales del servidor RADIUS (certificado y clave privada), además de la CRL, deben ser copiados a la ruta */etc/raddb/certs.* Tras ello, el módulo EAP declarado en el fichero */etc/raddb/eap.conf* debe ser ligeramente modificado. Las siguientes directivas deben quedar según se muestra a continuación:

```
default_eap_type = tls
# private_key_password = whatever
private_key_file = ${certdir}/radkey.pem
certificate_file = ${certdir}/radcrt.pem
CA_file = ${cadir}/crlca.pem
check_crl = yes
```

Como se quiere permitir exclusivamente el método de autenticación EAP-TLS se deben deshabilitar todos los módulos de la sección *Authorize* del servidor virtual habilitado por defecto, ubicado en */etc/raddb/sites-enabled/default*, a excepción de EAP:

```
authorize {
# chap
# mschap
# digest
# suffix
# files
# expiration
# logintime
eap {
ok = return
}
# pap
}
```

A continuación se genera el fichero diffie-helman (dh) y el aleatorio (random), para el establecimiento de sesiones tuneladas:

```
$ openssl dhparam -out dh 2048
$ openssl rand -out random 2048
```

Ahora se debe crear el fichero conjunto de CA y CRL, para que Freeradius pueda comprobar la lista de revocación:

```
$ cat cacrt.pem crl.pem > crlca.pem
```

El fichero */etc/raddb/clients.conf* debe contener una nueva declaración para permitir al autenticador (NAS) comunicarse con el servidor RADIUS:

```
client 192.168.1.1 {
```

```
secret =        testing123
nastype         = other
}
```

El autenticador también debe ser configurado para admitir exclusivamente autenticación 802.1X, basada en un servidor RADIUS externo, y compartiendo el secreto con el servidor:

```
Use 802.1X Authentication: YES
RADIUS Server Address: 192.168.1.100
RADIUS Server Port: 1812
RADIUS Shared Secret: testing123
```

En un conmutador Cisco Catalyst 2950 los comandos esenciales para realizar esta configuración son los siguientes:

```
# configure terminal
# aaa new-model
# aaa authentication dot1x default group radius
# interface fastethernet0/1
# dot1x port-control auto
# end
# radius-server host 192.168.1.100 auth-port 1812 key testing123
```

Ahora se puede levantar el servidor mediante el *daemon* radiusd, y este quedará a la escucha de mensajes *Access-Request* provenientes del NAS hacia el puerto de autenticación (UDP - 1812).

En el lado del cliente, el suplicante utilizado tiene un fichero de configuración que debe contener los parámetros de conexión. Se debe crear el archivo */etc/wpa_supplicant.conf* con el siguiente contenido:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
network={
key_mgmt=IEEE8021X
eap=TLS
ca_cert="/home/ignacio/ca/cacrt.pem"
client_cert="/home/ignacio/ca/clncrt.pem"
private_key="/home/ignacio/ca/clnkey.pem"
}
```

Una vez configurado el suplicante se puede realizar una conexión al autenticador mediante la siguiente instrucción:

```
$ wpa_supplicant -i rl0 -Dwired -c/etc/wpa_supplicant.conf
```

Si la firma es auténtica y el certificado vigente y no revocado, entonces el resultado debe ser similar al siguiente:

```
rl0: CTRL-EVENT-EAP-STARTED EAP authentication started
rl0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=13
rl0: CTRL-EVENT-EAP-METHOD EAP vendor 0 method 13 (TLS) selected
rl0: CTRL-EVENT-EAP-PEER-CERT depth=1 subject='/C=ES/O=SisCE/
OU=SecRootCA/emailAddress=sec@sisce.esc'
rl0: CTRL-EVENT-EAP-PEER-CERT depth=0 subject='/C=ES/O=SisCE/
OU=SecAUTH/CN=rad.sisce.esc'
rl0: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
```

En cualquier otra situación se denegará el acceso mostrándose la siguiente información de traza:

rl0: CTRL-EVENT-EAP-STARTED EAP authentication started

rl0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=13

rl0: CTRL-EVENT-EAP-METHOD EAP vendor 0 method 13 (TLS) selected

rl0: CTRL-EVENT-DISCONNECTED reason=3 locally_generated=1

Para este caso de denegación de servicio se ha utilizado un certificado expirado. En el registro del servidor RADIUS se puede observar la siguiente traza de intento infructuoso de conexión:

[eap] Request found, released from the list

[eap] EAP/tls

[eap] processing type tls

[tls] Authenticate

[tls] processing EAP-TLS

[tls] eaptls_verify returned 7

[tls] Done initial handshake

[tls] <<< TLS 1.0 Handshake [length 0a77], Certificate

[tls] chain-depth=1,

[tls] error=0

[tls] --> User-Name = SisCE-Worker

[tls] --> BUF-Name = ?

[tls] --> subject = /C=ES/O=SisCE/OU=SecRootCA
emailAddress=sec@sisce.esc

[tls] --> issuer = /C=ES/O=SisCE/OU=SecRootCA
emailAddress=sec@sisce.esc

[tls] --> verify return:1 --> verify error:num=10:certificate has expired

[tls] >>> TLS 1.0 Alert [length 0002], fatal certificate_expired

TLS Alert write:fatal:certificate expired

TLS_accept: error in SSLv3 read client certificate B

rlm_eap: SSL error error:140890B2:SSL routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned

SSL: SSL_read failed in a system call (-1), TLS session fails

TLS receive handshake failed during operation

[tls] eaptls_process returned 4

[eap] Handler failed in EAP/tls

[eap] Failed in EAP select

++[eap] returns invalid

Failed to authenticate the user

Login incorrect (certificate has expired): [SisCE-Worker/<via Auth-Type = EAP>] (from client 192.168.1.1 port 0 cli 001b9e05bd2a)

Using Post-Auth-Type REJECT

Devolviendo el sistema un mensaje *Access-Reject* al suplicante y rechazando así la conexión:

Sending Access-Reject of id 48 to 192.168.1.1 port 3072

EAP-Message = 0x04070004

Message-Authenticator = 0x00000000000000000000000000000000

Waking up in 1.1 seconds.

En cambio para una conexión exitosa el registro del servidor RADIUS muestra lo siguiente:

[eap] Request found, released from the list

[eap] EAP/tls

[eap] processing type tls

[tls] Authenticate

[tls] processing EAP-TLS

[tls] Received TLS ACK

[tls] ACK handshake is finished

[tls] eaptls_verify returned 3

[tls] eaptls_process returned 3

[tls] Adding user data to cached session

[eap] Freeing handler

++[eap] returns ok

Login OK: [SisCE-Worker/<via Auth-Type = EAP>] (from client 192.168.1.1 port 0 cli 001b9e05bd2a)

Finalmente el sistema devuelve un mensaje *Access-Accept* al suplicante, notificando así la aceptación de la conexión:

Sending Access-Accept of id 57 to 192.168.1.1 port 3072

MS-MPPE-Recv-Key = 0x861bc84a555656f9590b1e7fdbf4c79c3467983cce3f8cf671042f508db1ab7

MS-MPPE-Send-Key = 0xa1d4e004c38d478f84d840b9dad314163cdc5b162941682f4c3d2e8cbed4dd1

EAP-Message = 0x03080004

Message-Authenticator = 0x00000000000000000000000000000000

User-Name = "SisCE-Worker"

Finished request 8.

En este momento, tras la notificación de conexión, el NAS abre el puerto desautorizado (PAE) para iniciar el tráfico de datos entre el cliente y la red interna.

Para poder utilizar los credenciales contenidos en la tarjeta *hardware* en lugar de los almacenados en el disco duro de la computadora, es necesario modificar ligeramente el fichero */etc/wpa_supplicant.conf*:

```
ctrl_in terface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
pkcs11_engine_path=/usr/local/lib/engines/engine_pkcs11.so
pkcs11_module_path=/usr/local/lib/opensc-pkcs11.so
network={
key_mgmt=IEEE8021X
eap=TLS
identity="SisCE-Worker"
# use engine for private key operations (OpenSSL specific)
engine=1
# engine id string (OpenSSL specific)
engine_id="pkcs11" # the private key's id when using engine (OpenSSL specific)
key_id="slot_1-id_51f191e2aa08d518a5f91620e7562a32a936545e"
# the certificate's id when using engine
cert_id="slot_1-id_51f191e2aa08d518a5f91620e7562a32a936545e"
# the CA certificate's id when using engine
ca_cert_id="slot_1-id_of37e5fca6e04e356d41b0193d725eb0e54fc3bd"
}
```

El formato de referencia a la tarjeta es *slot_X-id_ID*, donde X es el número de la ranura de inserción del lector, e ID el identificador del credencial específico contenido en la tarjeta. Ambos pueden ser obtenidos para cada uno de los credenciales ejecutando las siguientes instrucciones:

```
$ pkcs15-tool --list-certificates
$ pkcs15-tool --list-keys
$ pkcs11-tool --list-slots --module opensc-pkcs11.so
```

Ahora se puede ejecutar de nuevo el suplicante (wpa_supplicant), de igual manera a como se hizo anteriormente, pero en esta ocasión como *daemon*, y a continuación se introduce el PIN de paso de la tarjeta haciendo uso del interfaz de línea de comandos wpa_cli:

```
$ wpa_supplicant -i rl0 -Dwired -c/etc/wpa_supplicant.conf -B
$ wpa_cli PIN 0 1234
```

5.4 Anexión de acceso inalámbrico

Para la puesta en funcionamiento de la infraestructura Wi-Fi es necesario añadir el AP a la configuración del servidor RADIUS, para que sea confiable y ambos puedan comunicarse. Igualmente debe configurarse el AP para aceptar y securizar el tráfico de datos desde el suplicante, siendo también necesario realizar algún cambio en la configuración de este último.

En el archivo de configuración */etc/wpa_supplicant.conf* se añaden o comentan (deshabilitar) las siguientes líneas:

```
ssid=”one”
# key_mgmt=IEEE8021X
key_mgmt=WPA-EAP
proto=RSN
pairwise=CCMP
```

group=CCMP

Con estas directivas se anexa al protocolo EAP la gestión de clave (PMK) y cifrado WPA2 en modo *Enterprise*. Estableciendo exclusivamente una infraestructura RSN (Robust Security Network).

El punto de acceso (AP) debe presentar una configuración similar a la siguiente:

Security Mode:	WPA2-Enterprise
WPA Algorithms:	AES
RADIUS Server Address:	192.168.1.100
RADIUS Server Port:	1812
RADIUS Shared Secret:	testing124
Key Renewal Interval:	600

Los dos primeros valores son específicos para la seguridad de la infraestructura, y aunque están indicados de distinta forma, son equivalentes a los seleccionados para la configuración del suplicante.

Para establecer esta configuración en un *router* Comtrend Gigabit 802.11n/g/b, la instrucción esencial es la siguiente:

```
# wlan security wpa2 --wpaenc aes --rekey 600 --rasip 192.168.1.100 \
--raspt 1812 --raskey testing124
```

En el fichero */etc/raddb/clients.conf* es necesario añadir una nueva declaración para que el AP pueda comunicarse con el servidor RADIUS:

```
client 192.168.1.2 {
```

```
secret = testing124
nastype = other
}
```

Ahora puede ejecutarse el suplicante, especificando el driver e interfaz *wireless* que se van a utilizar, y los resultados son muy similares a los expuestos en el apartado anterior:

```
$ wpa_supplicant -i wlan0 -D bsd -c /etc/wpa_supplicant.conf -B
$ wpa_cli PIN 0 1234
```

5.5 Anexión de acceso remoto

Para este tipo de acceso se emplea el software OpenVPN, que permite levantar y mantener enlaces remotos sobre la capa 3 del modelo OSI, con gran simplicidad y eficacia. Es una herramienta bastante flexible y su rendimiento, siendo una aplicación de espacio de usuario, no es despreciable frente a implementaciones que se integran con el *kernel* del sistema operativo. Por estas y otras razones, es un software adecuado para implementar un *road warrior* (punto de acceso remoto accesible mediante una red pública).

La mayor parte del *outsourcing* de servicios (externalización de servicios) del sector empresarial suele operarse mediante enlaces remotos que el cliente pone a disposición del proveedor. En este sentido existen dos enfoques a la hora de plantear el diseño de la infraestructura, con sus correspondientes matices e inconvenientes:

1. Todo el tráfico de datos del empleado es canalizado a través del enlace VPN, lo cual genera una mayor carga sobre la red interna del cliente, disminuyendo así, proporcionalmente, su rendimiento. Esta implementación permite tener un mayor control sobre la información que maneja el empleado, pero por el contrario requiere una configuración algo más compleja, tanto del lado del cliente como del servidor.

2. Del tráfico de datos del empleado, sólo aquel que está destinado a la red interna del cliente (red remota), es canalizado a través del enlace VPN, mientras que el resto es encaminado a través del sistema local del empleado. Con este enfoque la configuración es más simple en ambos lados, y la carga de datos sobre la red del cliente es menor. Esta implementación permite al empleado realizar acciones sobre Internet (se entiende necesarias para el correcto desempeño de su actividad laboral) que conllevan un gran tráfico de datos, tales como *streaming* o descarga de archivos grandes, sin pasar a través de la red del cliente.

Para configurar el servidor, en primer lugar se crea el directorio */etc/openvpn*, donde se deben colocar todos los elementos necesarios para la correcta ejecución del servicio. Se copia el archivo de ejemplo */etc/usr/local/share/examples/openvpn/sample-config-files/server.conf* a la ubicación previamente mencionada, y se modifican los siguientes valores:

```
ca /etc/openvpn/cacrt.pem
cert /etc/openvpn/tuncrt.pem
```

```
key /etc/openvpn/tunkey.pem                 # This file should be kept secret
dh /etc/openvpn/dh
ifconfig-pool-persist /etc/openvpn/ipp.txt
cipher BF-CBC                               # Blowfish (default)
```

Como se ha indicado en la configuración es necesario copiar al directorio */etc/openvpn* los archivos *cacrt.pem*, *tuncrt.pem* y *tunkey.pem* (creados en el apartado 3.3), e igualmente en el mismo directorio se genera un nuevo archivo diffie-hellman:

```
$ openssl dhparam -out dh 2048
```

A continuación se procede de manera similar con la configuración del cliente. Se copia el fichero */usr/local/share/examples/openvpn/sample-config-files/client.conf* al directorio */etc/openvpn* (creado previamente), y se modifican los siguientes valores:

```
remote tun.sisce.esc 1194
user nobody
group nobody
ca /etc/openvpn/cacrt.pem
cert /etc/openvpn/clncrt.pem
key /etc/openvpn/clnkey.pem
cipher BF-CBC
```

Ahora también los archivos referenciados por el cliente deben ser copiados al directorio */etc/openvpn.*

La configuración que se ha establecido por defecto en base a los ficheros de ejemplo se corresponde con el tunelamiento exclusivo de los datos destinados a la red remota. Para canalizar todo el tráfico de red a través del túnel es necesario además habilitar la siguiente directiva en el servidor:

```
push "redirect-gateway def1"
```

Pero como se ha mencionado este enfoque conlleva algo más de configuración manual en ambos lados, dado que algunos servicios locales al empleado como DHCP no deben ser tunelizados, y el DNS e Internet deben ser implantados vía proxy en la red remota.

Ahora se pueden ejecutar tanto el servidor como el cliente con sendas siguientes instrucciones, respectivamente:

```
$ openvpn --config /etc/openvpn/server.conf
$ openvpn --config /etc/openvpn/client.conf
```

Tras una conexión exitosa del cliente la salida estándar del servidor muestra la siguiente información de traza:

```
Fri Jun  6 11:08:10 2014 Initialization Sequence Completed
Fri Jun  6 11:08:30 2014 192.168.0.4:36250 TLS: Initial packet from
[AF_INET]192.168.0.4:36250, sid=398686d9 2efe22ab
Fri Jun  6 11:08:30 2014 192.168.0.4:36250 VERIFY OK: depth=1, C=ES,
O=SisCE, OU=SecRootCA, emailAddress=sec@sisce.esc
Fri Jun  6 11:08:30 2014 192.168.0.4:36250 VERIFY OK: depth=0, C=ES,
O=SisCE, OU=Preventa, CN=JIMENEZ PINTO, IGNACIO
Fri Jun  6 11:08:30 2014 192.168.0.4:36250 Data Channel Encrypt: Cipher 'BF-
CBC' initialized with 128 bit key
Fri Jun  6 11:08:30 2014 192.168.0.4:36250 Data Channel Encrypt: Using 160 bit
message hash 'SHA1' for HMAC authentication
```

Fri Jun 6 11:08:30 2014 192.168.0.4:36250 Data Channel Decrypt: Cipher 'BF-CBC' initialized with 128 bit key

Fri Jun 6 11:08:30 2014 192.168.0.4:36250 Data Channel Decrypt: Using 160 bit message hash 'SHA1' for HMAC authentication

Fri Jun 6 11:08:30 2014 192.168.0.4:36250 Control Channel: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-AES256-SHA, 2048 bit RSA

Fri Jun 6 11:08:30 2014 192.168.0.4:36250 [JIMENEZ PINTO, IGNACIO] Peer Connection Initiated with [AF_INET]192.168.0.4:36250

Fri Jun 6 11:08:30 2014 JIMENEZ PINTO, IGNACIO/192.168.0.4:36250 MULTI_sva: pool returned IPv4=10.8.0.6, IPv6=(Not enabled)

Fri Jun 6 11:08:30 2014 JIMENEZ PINTO, IGNACIO/192.168.0.4:36250 MULTI: Learn: 10.8.0.6 -> JIMENEZ PINTO, IGNACIO/192.168.0.4:36250

Fri Jun 6 11:08:30 2014 JIMENEZ PINTO, IGNACIO/192.168.0.4:36250 MULTI: primary virtual IP for JIMENEZ PINTO, IGNACIO/192.168.0.4:36250: 10.8.0.6

Fri Jun 6 11:08:32 2014 JIMENEZ PINTO, IGNACIO/192.168.0.4:36250 PUSH: Received control message: 'PUSH_REQUEST'

Fri Jun 6 11:08:32 2014 JIMENEZ PINTO, IGNACIO/192.168.0.4:36250 send_push_reply(): safe_cap=940

Fri Jun 6 11:08:32 2014 JIMENEZ PINTO, IGNACIO/192.168.0.4:36250 SENT CONTROL [JIMENEZ PINTO, IGNACIO]: 'PUSH_REPLY,route 10.8.0.1,topology net30,ping 10,ping-restart 120,ifconfig 10.8.0.6 10.8.0.5' (status=1

En el lado del cliente la información de conexión exitosa es similar a la anterior:

Fri Jun 6 09:10:00 2014 TLS: Initial packet from [AF_INET]192.168.0.3:1194, sid=80276a61 6d6d9b4d

Fri Jun 6 09:10:00 2014 VERIFY OK: depth=1, C=ES, O=SisCE, OU=SecRootCA, emailAddress=sec@sisce.esc

Fri Jun 6 09:10:00 2014 VERIFY OK: nsCertType=SERVER

Fri Jun 6 09:10:00 2014 VERIFY OK: depth=0, C=ES, O=SisCE, OU=SecVPN, CN=tun.sisce.esc

Fri Jun 6 09:10:00 2014 Data Channel Encrypt: Cipher 'BF-CBC' initialized with 128 bit key

Fri Jun 6 09:10:00 2014 Data Channel Encrypt: Using 160 bit message hash

'SHA1' for HMAC authentication

Fri Jun 6 09:10:00 2014 Data Channel Decrypt: Cipher 'BF-CBC' initialized with 128 bit key

Fri Jun 6 09:10:00 2014 Data Channel Decrypt: Using 160 bit message hash 'SHA1' for HMAC authentication

Fri Jun 6 09:10:00 2014 Control Channel: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-AES256-SHA, 2048 bit RSA

Fri Jun 6 09:10:00 2014 [tun.sisce.esc] Peer Connection Initiated with [AF_INET]192.168.0.3:1194

Fri Jun 6 09:10:02 2014 SENT CONTROL [tun.sisce.esc]:'PUSH_REQUEST' (status=1)

Fri Jun 6 09:10:02 2014 PUSH: Received control message:'PUSH_REPLY,route 10.8.0.1,topology net30,ping 10,ping-restart 120,ifconfig 10.8.0.6 10.8.0.5'

Fri Jun 6 09:10:02 2014 OPTIONS IMPORT: timers and/or timeouts modified

Fri Jun 6 09:10:02 2014 OPTIONS IMPORT: --ifconfig/up options modified

Fri Jun 6 09:10:02 2014 OPTIONS IMPORT: route options modified

Fri Jun 6 09:10:02 2014 ROUTE_GATEWAY 192.168.0.1

Fri Jun 6 09:10:02 2014 TUN/TAP device /dev/tun0 opened

Fri Jun 6 09:10:02 2014 do_ifconfig, tt-> ipv6=0, tt-> did_ifconfig_ipv6_setup=0

Fri Jun 6 09:10:02 2014 /sbin/ifconfig tun0 10.8.0.6 10.8.0.5 mtu 1500 netmask 255.255.255.255 up

Fri Jun 6 09:10:02 2014 /sbin/route add -net 192.168.1.0 10.8.0.5 255.255.255.0 add net 192.168.1.0: gateway 10.8.0.5

Fri Jun 6 09:10:02 2014 /sbin/route add -net 10.8.0.1 10.8.0.5 255.255.255.255 add net 10.8.0.1: gateway 10.8.0.5

Fri Jun 6 09:10:02 2014 GID set to nobody

Fri Jun 6 09:10:02 2014 UID set to nobody

Fri Jun 6 09:10:02 2014 Initialization Sequence Completed

En las trazas anteriores se puede observar como el servidor muestra la cadena de texto “Initialization Sequence Completed” justo al principio, a

diferencia del cliente que lo hace al final, lo cual refleja en ambos casos el éxito de la activación con la parametrización fijada.

A continuación y al igual que se realizó con el servidor RADIUS, es necesario definir la CRL para poder denegar intentos de conexión que no cumplen con los requisitos deseados. Esta CRL a diferencia de la de FreeRADIUS, permite una actualización sin reinicio, y consecuentemente los empleados pueden seguir conectados durante el proceso. Además de lo mencionado también permite un mantenimiento sencillo, según el cual es suficiente referenciar el número de serie de un certificado en un directorio previamente definido para que quede revocado, y por lo tanto su acceso en adelante sea denegado. Pero para mantener el sincronismo entre ambos servicios se mantiene el enfoque clásico, con la salvedad, de que no es necesario fusionar el certificado de la CA con el de la CRL. En consecuencia en el fichero */etc/openvpn/server.conf* se añade la siguiente directiva:

```
crl-verify /etc/openvpn/crl.pem
```

Tras copiar la CRL al directorio referenciado y probar el resultado con un certificado de cliente revocado, la información que se puede visualizar en el servidor es la siguiente:

```
Fri Jun  6 13:18:28 2014 Initialization Sequence Completed

Fri Jun  6 13:18:35 2014 192.168.0.4:33624 TLS: Initial packet from
[AF_INET]192.168.0.4:33624, sid=5b1f7f91 fa2b06ad

Fri Jun  6 13:18:36 2014 192.168.0.4:33624 CRL CHECK OK: C=ES, O=SisCE,
OU=SecRootCA, emailAddress=sec@sisce.esc

Fri Jun  6 13:18:36 2014 192.168.0.4:33624 VERIFY OK: depth=1, C=ES,
```

O=SisCE, OU=SecRootCA, emailAddress=sec@sisce.esc

Fri Jun 6 13:18:36 2014 192.168.0.4:33624 CRL CHECK FAILED: C=ES, O=SisCE, OU=Finanzas, CN=NATIVIDAD PINTO, MARIA is REVOKED

Fri Jun 6 13:18:36 2014 192.168.0.4:33624 TLS_ERROR: BIO read tls_read_plaintext error: error:140890B2:SSL routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned

Fri Jun 6 13:18:36 2014 192.168.0.4:33624 TLS Error: TLS object -> incoming plaintext read error

Fri Jun 6 13:18:36 2014 192.168.0.4:33624 TLS Error: TLS handshake failed

De la traza anterior se deduce que el intento de conexión ha sido denegado debido a la utilización de un certificado revocado por parte del empleado.

El despliegue realizado hasta el momento sólo encamina paquetes entre máquinas que se encuentran situadas dentro del segmento de red privado y virtual (10.8.0/24), el cual ha creado el software OpenVPN. Pero lo habitual es que la red interna de la compañía (red remota) esté igualmente segmentada, y el empleado necesite alcanzar máquinas o sistemas pertenecientes a otros segmentos de red (sub-redes). Para poder realizar estas acciones, es necesario que el servidor anuncie al cliente (durante el establecimiento de la conexión) las sub-redes privadas que este último puede alcanzar. Esta característica puede ser personalizada por colectivo o incluso individualizada, permitiendo distintos perfiles de acceso.

Para permitir a los empleados acceso a segmentos de red internos deben declararse en la configuración del servidor directivas *push,* como en el siguiente ejemplo:

push "route 192.168.1.0 255.255.255.0"

Esto anuncia al cliente que el empleado dispone de la posibilidad de encaminar tráfico de datos hacia ese segmento de red. Pero igualmente y dado que los paquetes llevan direcciones fuente pertenecientes a la red privada virtual, los dispositivos encaminadores internos deben disponer de rutas de red que permitan devolver los paquetes de respuesta al servidor, para que este último pueda encaminarlos hacia el *host* del empleado, y por idéntica razón, el servidor ha de activar en su *kernel* la funcionalidad de encaminamiento entre redes (net.inet.ip.forwarding), que le permite actuar como un *router*.

En un entorno de producción también es habitual que el servidor VPN se encuentre detrás de un *appliance* que implemente un *firewall,* y realice las funciones de punto de acceso en contacto directo con la red pública (Internet). Bajo esta circunstancia, es necesario que dicho dispositivo abra el puerto en el cual el servidor VPN permanece a la escucha (UDP - 1194, por defecto), y lo direccione hacia este último (NAT).

Para poder hacer uso de la *smart card* es necesario deshabilitar y añadir las siguientes directivas en la configuración del cliente:

```
# cert /etc/openvpn/clncrt.pem

# key /etc/openvpn/clnkey.pem

management localhost 7505

management-query-passwords

pkcs11-providers /usr/local/lib/opensc-pkcs11.so

pkcs11-id 'EnterSafe/PKCS\x2315/1320463413190613/OpenSC\x20Card
\x20\x28User\x20PIN\x29/31'
```

El identificador completo de credencial puede ser obtenido como el "Serialized id" mediante la siguiente instrucción:

```
$ openvpn --show-pkcs11-ids /usr/local/lib/opensc-pkcs11.so
```

Ahora se puede ejecutar de nuevo el cliente, pero esta vez como *daemon,* y tras ello introducir el PIN de la tarjeta mediante el interfaz de gestión:

```
$ openvpn --daemon --config /etc/openvpn/client.conf
$ echo "password 'OpenSC Card (User PIN) token' '1234'" | \
telnet localhost 7505
```

El valor correcto para el *string* "OpenSC Card (User PIN)" puede ser obtenido del atributo "token label", en la salida mostrada por la ejecución del siguiente comando:

```
$ pkcs11-tool --list-slots --module opensc-pkcs11.so
```

El resultado visualizado es prácticamente idéntico al mencionado con anterioridad, con la única diferencia de que en esta ocasión aparece un *prompt* de solicitud de contraseña, para introducir el PIN de la tarjeta, pero el *daemon* inmediatamente recoge la información del interfaz de gestión:

```
Enter OpenSC Card (User PIN) token Password:
```

En caso de no hacer uso del interfaz de gestión se debe introducir interactivamente el PIN, tal y como ocurre con el interfaz wpa_cli para el acceso inalámbrico.

5.6 Mejoras al sistema global

En los apartados anteriores se ha expuesto el diseño e implantación del sistema de control de acceso planteado inicialmente, sin entrar en cuestiones de integración con elementos comunes a toda infraestructura corporativa (servicios WEB, SQL o LDAP), e igualmente se han omitido detalles de *hardening* (fortalecimiento del sistema) y *tunning* (rendimiento del sistema). Todos estos temas son adecuadamente valorados en infraestructuras avanzadas de mayor complejidad, no siendo, por consiguiente, objetivo del presente texto. No obstante se plantean aquí una serie de posibles mejoras relacionadas con estas u otras cuestiones, que pueden servir como punto de partida para posteriormente profundizar en la extensión y mejora del diseño planteado:

- Utilizar el protocolo EAP-TTLS que implementa un doble proceso de autenticación mediante la utilización de dos túneles TLS. El empleo de certificado de cliente puede forzarse añadiendo la directiva *EAP-TLS-Require-Client-Cert* a la lista de atributos de control de FreeRadius, en la sección authorize del servidor virtual:

```
update control {
EAP-TLS-Require-Client-Cert = Yes
}
```

  El primer túnel se establece para el intercambio de credenciales haciendo uso de métodos de clave compartida (PAP, CHAP), mientras que el segundo sirve para intercambiar claves de cifrado, las cuales son utilizadas posteriormente para el tunelamiento de la sesión de

usuario (WPA2-RSN). A diferencia de EAP-TLS aquí interviene en la autenticación una clave compartida (como elemento extra), la cual puede estar almacenada en algún *data source* corporativo (BBDD, LDAP), e igualmente residir en el PC del empleado, obligando así a un suplantador que logre apoderarse de una *smart card* corporativa y de su PIN asociado, a necesitar acceder a un PC de empleado para substraer la clave compartida.

Este esquema puede ser igualmente invertido, manteniendo credenciales simples para los empleados (usuarios) y certificados PKI para sus equipos (ordenadores). Ambos tipos de credencial pueden ser personalizados por usuario y/o equipo, o generalizados por sector departamental (grupo o colectivo).

- Adquirir tarjetas criptográficas que soporten además de un PIN de usuario para lectura un *Security Officer PIN* (global) para escritura, que no es exactamente el caso de la tarjeta hardware utilizada para la implementación de este proyecto, la cual dispone de un PIN único que realiza ambas funciones (o casi, pues no protege del formateado de la misma). Al disponer de un SO-PIN independiente, este podría quedar relegado al conocimiento y uso exclusivo de la persona responsable del servicio, evitando así cualquier escritura o borrado de objetos en la tarjeta por parte del empleado (e incluso el formateado de la misma).

- Existe una posibilidad (muy remota) de efectuar un ataque MitM (man-in-the-middle) contra el sistema, exclusivamente para el acceso

cableado y el inalámbrico, y especialmente en este último por la cobertura que ofrece. Para ello un atacante tendría que conseguir los credenciales del servidor (certificado y clave privada), y después suplantarle ante un cliente. Esto es posible debido a que algunos suplicantes actuales no implementan comprobación de CRL, ni soporte OCSP.

Hacerse con los credenciales del servidor de autenticación supondría tener o haber tenido acceso al mismo, luego si ya se hubiera comprometido la red… ¿para qué realizar un ataque MitM?, el cual, en teoría, tiene la misma finalidad. Se puede pensar en un empleado descontento o en espionaje corporativo, cualquier opción es válida.

La mejor contramedida posible ante este hecho es realizar una verificación posterior a la conexión, haciendo disponible el certificado del servidor en una ubicación accesible al cliente, y permitiendo así la desconexión inmediata si este último detecta alguna anomalía durante la verificación.

- Para fortalecer los servidores se pueden ejecutar sendos *daemons* en jaulas, habilitando directivas *chroot* en los ficheros de configuración (radiusd.conf y server.conf). Esto imposibilita que el *daemon* acceda a otras partes del sistema de ficheros, relegándolo exclusivamente al directorio definido en la directiva. Esto conlleva que deban copiarse a dicho directorio todos los archivos a los cuales el *daemon* requiera tener acceso tras su ejecución, y para su correcto funcionamiento, claves, certificados, CRL, etc.

La mejora proporcionada por una *chroot jail* desde el punto de vista de la seguridad es ante un *exploit* de inserción de código ejecutado sobre el *daemon,* pues esta no permite que el *daemon* sea capaz de poner en riesgo el resto del sistema, salvo su propio despliegue parcial.

- Recursos internos como un servicio de Directorio Activo o una Base de Datos relacional (OpenLDAP o PostgreSQL). El diseño del control de acceso al medio basado en la integración con estos u otros recursos internos no es estrictamente considerado una mejora de seguridad, pero para algunas corporaciones esta integración puede suponer un requisito técnico e igualmente una necesidad de funcionalidad. Estas *data sources* pueden almacenar toda la información del empleado, desde sus datos de residencia y cuenta bancaria hasta simples credenciales de acceso o certificados PKI, los cuales pueden ser revocados o validados, y accedidos para su comprobación desde sistemas de autenticación como RADIUS, a través de módulos de software (rlm_sql_postgresql.so ó rlm_ldap.so).

  Otro de los usos más habituales de este tipo de recursos es su implicación en la fase de autorización del proceso de acceso, pues suelen almacenar atributos específicos de empleado y NAS, que son asignados durante el proceso y utilizados tras la concesión del acceso, como el rango de utilización horaria, el segmento específico de red al que tener acceso, el puerto de conexión, o incluso el tipo de tunelización.

- Si se utiliza la rama 2.2.x de FreeRADIUS y debido a que el protocolo no tiene soporte criptográfico ni permite la integración con tales *suites*, se pueden fortalecer todos los enlaces internos haciendo uso del protocolo IPsec, el cual es relativamente fácil de implementar sobre entornos de direccionamiento estático y con sistemas que lo integran de serie. En cambio si se actualiza a la versión 3 del servidor se puede utilizar una de las mejoras incorporadas, el protocolo RadSec, que implementa RADIUS sobre TLS, pero igualmente todos los clientes del servidor deben soportar el protocolo. Esto realmente mejora la seguridad, especialmente entre dominios administrativos enlazados por redes públicas, o potencialmente hostiles.

- La denegación de servicio (DoS - Denial of Service) puede producirse por un ataque directo contra la infraestructura o por una sobrecarga involuntaria de la misma. El servidor FreeRADIUS permite afinar la protección contra este problema mediante la modificación de las directivas *max_attributes*, *reject_delay* o/y *cleanup_delay*. Por su parte el servidor OpenVPN puede incrementar su protección mediante el uso de la directiva *tls-auth,* que emplea una clave compartida por ambos extremos para añadir una firma HMAC a todos los mensajes SSL/TLS intercambiados.

  El mecanismo utilizado por el servidor VPN protege además frente a otras amenazas, como el escaneo de puertos, o algunas vulnerabilidades que pueda manifestar el protocolo SSL/TLS, pero por el contrario requiere implementación y mantenimiento en ambos extremos. También es cierto que aunque el servidor de autenticación

no dispone de estos mecanismos de mitigación no está expuesto a la red pública, salvo en muy menor medida a la cobertura Wi-Fi.

- Se puede utilizar un *firewall* integrado por cada servidor, que filtre los paquetes que llegan de manera más precisa, saneándolos y protegiendo contra posibles amenazas internas. Algunas directivas del Packet Filter (PF) como *block-policy*, *scrub* y/o *antispoof,* pueden evitar reintentos en conexiones fallidas, desechar paquetes malformados, y bloquear paquetes proviniendo u originándose en interfaces y direcciones que no son posibles a nivel lógico.

  Por otra parte y aunque el servidor Radius ignora todos los intentos de conexión que no provengan de clientes confiables (debido al secreto compartido), se pueden afinar reglas como las siguientes para tener un mayor control del tráfico gestionado:

```
udp = "{ radius }"
if = “xl0“
block all
pass in on $if inet proto udp from $localnet to $address port $udp keep state
pass out on $ext_if inet proto udp from $address to $localnet port $udp keep state
```

  Con estas reglas se pretende discriminar entre interfaces/direcciones de servicio o de gestión, orígenes y destinos, e igualmente protocolos. Incluso en un enfoque más extremista se podrían utilizar *overload tables* (tablas de sobrecarga), para tratar de aislar temporalmente a iniciadores de degradación de servicio.

- También es importante anotar que el dimensionamiento del volumen de usuarios de una infraestructura Radius debe ser realizado concienzudamente. Para ello la directiva *max_requests* que está relacionada directamente con dicho volumen debe ser ajustada proporcional y convenientemente (*tunning* del servicio).

5.7 Infraestructura recomendada

Una vez planteado y expuesto el diseño inicial, y habiendo resaltado una serie de puntos de mejora, se puede afrontar el diseño de una arquitectura técnicamente más compleja y avanzada, como la que se muestra en la siguiente figura:

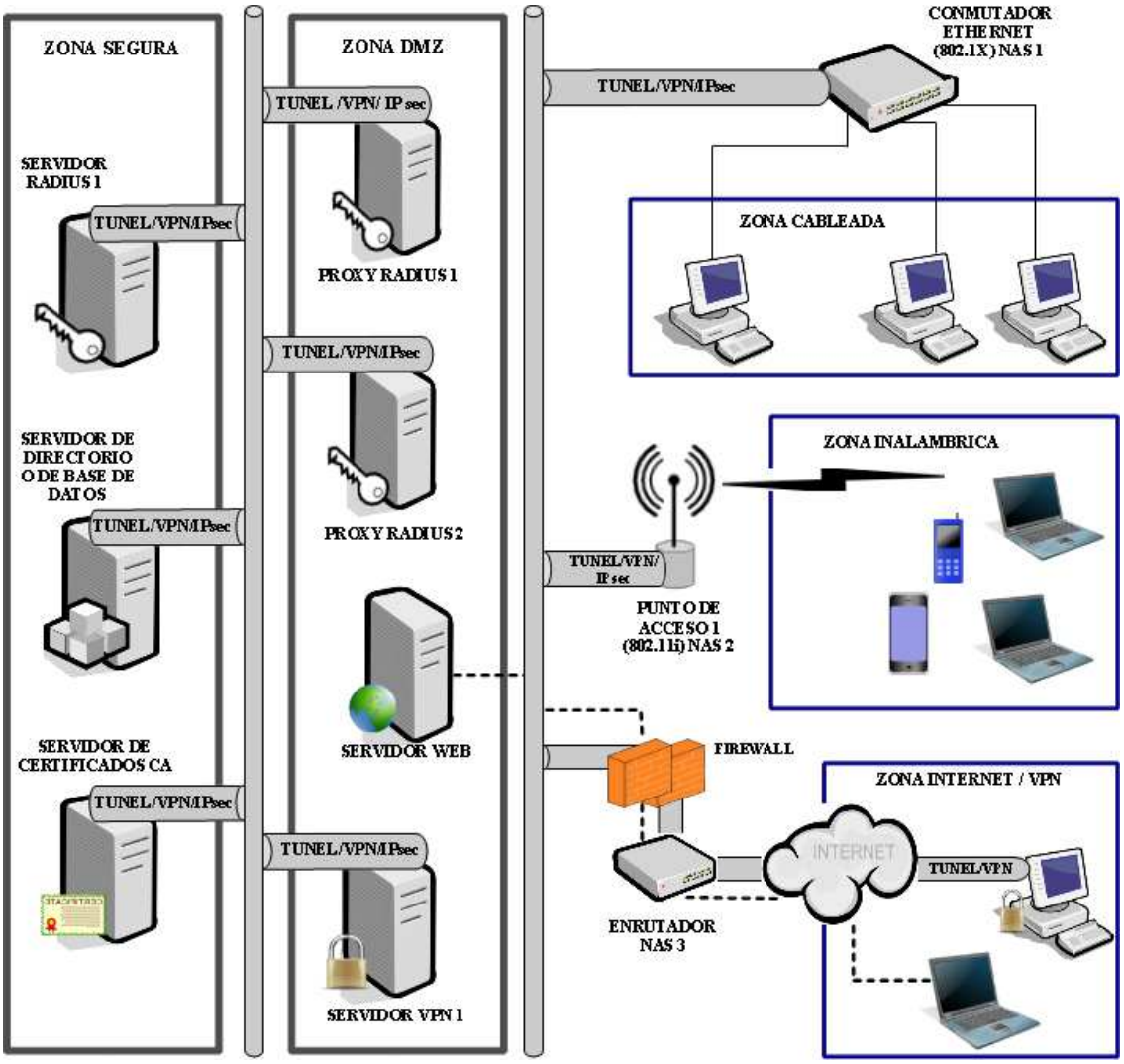

Figura 15.- Infraestructura recomendada para el control de acceso global.

La arquitectura de la figura 15 consta de tres capas o niveles. Desde el más interno hasta el más externo:

- Zona segura > Información sensible.
- Zona DMZ > Sistemas frontales.
- Zona perimetral > Puntos de acceso.

Todos los enlaces internos están tunelizados mediante IPsec o algún otro software criptográfico de VPN. En el perímetro de la infraestructura los

acoplamientos a la misma están controlados por protocolos de la capa de enlace, mediante la implantación del estándar 802.1x en todos los autenticadores (equipos NAS). El único servicio que se escapa de este control exhaustivo y tunelamiento es el servicio WEB, que como es frecuente suele disponer de un perfil público, para que la compañía pueda ejercer su actividad profesional convenientemente.

La arquitectura Radius está jerarquizada mediante el empleo de sistemas *proxy* que consultan al servidor maestro, y que disponen de un par de NICs para ser los únicos sistemas que tienen acceso a ambas zonas (Segura y DMZ).

El servidor de certificados presentado al inicio del capítulo como un sistema portable se integra aquí con la infraestructura, pues a muchas organizaciones no les parecerá una buena decisión de diseño ni una opción confiable mantenerlo aislado y no visible, o incluso bajo algunas circunstancias puede ser deseable para la compañía disponer de un servicio automatizado de certificación.

Para concluir, cualquier sistema puede ser convenientemente replicado para proporcionar sus servicios en alta disponibilidad y con balanceo de carga, aportando así mayor robustez y estabilidad a la infraestructura.

6. PRUEBAS

6. Pruebas

En este capítulo se detalla la ejecución de un *test plan* básico con pruebas integradas, pruebas *hop-to-hop* y *end-to-end,* para verificar el comportamiento correcto de cada componente, e igualmente la integración entre los mismos. Son pruebas significativas de cara a la puesta en producción del sistema.

Los casos de uso que constituyen el plan de pruebas son los siguientes:

- Comprobar tarjeta de identificación electrónica.
- Verificar configuración del servidor RADIUS.
- Conexión cableada con mecanismo EAP-MD5.
- Acceso cableado/inalámbrico con método EAP-TLS.
- Verificar configuración del servidor VPN.
- Acceso remoto con método UDP-TLS.
- Verificar la conexión a la LAN mediante ICMP.

Para cada uno de los casos de uso citados sólo se mostrará la información relevante cuando esta sea muy extensa.

Comprobar tarjeta de identificación electrónica

Tras la ejecución de los comandos del apartado 5.2 (Creación de certificados digitales) se comprueba que la tarjeta contiene la información adecuada con la siguiente instrucción:

```
$ pkcs15-tool --dump

Private RSA Key [JIMENEZ PINTO, IGNACIO]
Object Flags : [0x3], private, modifiable
Usage : [0xC], sign, signRecover
Access Flags : [0xD], sensitive, alwaysSensitive, neverExtract
ModLength : 2048
Key ref : 1 (0x1)
Native : yes
Path : 3f005015
Auth ID : 01
ID : 51f191e2aa08d518a5f91620e7562a32a936545e
GUID : {e98fc46c-1940-9c91-dc09-4463ade981c0}

X.509 Certificate [JIMENEZ PINTO, IGNACIO]
Object Flags : [0x2], modifiable
Authority : no
Path : 3f0050153100
ID : 51f191e2aa08d518a5f91620e7562a32a936545e
GUID : {e98fc46c-1940-9c91-dc09-4463ade981c0}
Encoded serial : 02 01 03
```

X.509 Certificate [/C=ES/O=SisCE/OU=SecRootCA/emailAddress=sec@sisce.esc]

Object Flags : [0x2], modifiable

Authority : yes

Path : 3f0050153101

ID : b2cb78876f0f01b195917eec0db98dc0b68268bf

GUID : {d80011ae-de9c-5994-d639-0db2a4ed9574}

Encoded serial : 02 01 00

Como se puede observar la tarjeta contiene dos certificados digitales extraíbles y una clave privada no extraíble (neverExtract), esta última es usable únicamente para la firma digital y su verificación (sign, signRecover), pero no para el cifrado de información (confidencialidad).

Ahora se puede comprobar que OpenSSL es capaz de operar con los módulos criptográficos para el uso de la tarjeta (opensc-pkcs11.so y engine_pkcs11.so), interactuando así con las aplicaciones de usuario. Para ello se ejecuta openssl y en su *prompt* se introduce la secuencia mostrada a continuación:

$ openssl

OpenSSL> engine -t dynamic –pre \

SO_PATH:/usr/local/lib/engines/engine_pkcs11.so \

-pre ID:pkcs11 \

-pre LIST_ADD:1 \

-pre LOAD \

-pre MODULE_PATH: /usr/local/lib/opensc-pkcs11.so

(dynamic) Dynamic engine loading support

[Success]: SO_PATH: /usr/local/lib/engines/engine_pkcs11.so

[Success]: ID: pkcs11

[Success]: LIST_ADD: 1

[Success]: LOAD

[Success]: MODULE_PATH:/usr/local/lib/opensc-pkcs11.so

Loaded: (pkcs11) pkcs11 engine

[available]

La *suite* criptográfica responde con el *string* "available", indicando con ello que tiene soporte y es capaz de manejar los módulos indicados (engines).

Finalmente y para comprobar la funcionalidad de la tarjeta, al mismo tiempo que se prueba la integración de esta última con el lector y el controlador del sistema, se puede realizar un *test* completo:

$ pkcs11-tool --login --test --module /usr/local/lib/opensc-pkcs11.so

Digests:

all 4 digest functions seem to work

MD5: OK

SHA-1: OK

RIPEMD160: OK

Signatures (currently only RSA signatures)

testing key 0 (JIMENEZ PINTO, IGNACIO)

all 4 signature functions seem to work

testing signature mechanisms:

RSA-X-509: OK

RSA-PKCS: OK

SHA1-RSA-PKCS: OK

MD5-RSA-PKCS: OK

RIPEMD160-RSA-PKCS: OK

SHA256-RSA-PKCS: OK

Verify (currently only for RSA):

testing key 0 (JIMENEZ PINTO, IGNACIO)

RSA-X-509: OK

RSA-PKCS: OK

SHA1-RSA-PKCS: OK

MD5-RSA-PKCS: OK

RIPEMD160-RSA-PKCS: OK

Unwrap: not implemented

Decryption (RSA)

testing key 0 (JIMENEZ PINTO, IGNACIO) -- can't be used to decrypt, skipping

No errors

El sistema va realizando las pruebas y mostrando confirmaciones, tanto para la firma como para la comprobación de la misma, y con todos los algoritmos soportados.

Es muy importante verificar las especificaciones técnicas del hardware tanto como sea posible, porque si no es completamente compatible con los estándares pueden visualizarse mensajes de error como los siguientes (en pkcs11-tool y en openssl, respectivamente):

error: PKCS11 function C_Login failed: rv = CKR_GENERAL_ERROR (0x5)

```
3074315964:error:80005005:Vendor defined:PKCS11_login:General
Error:p11_slot.c:157:
```

Verificar configuración del servidor RADIUS

Tras la ejecución de los comandos del apartado 5.3 (Despliegue de acceso cableado) para configurar el servicio de autenticación, se comprueba la correcta configuración de este último ejecutando el *daemon* de FreeRadius en modo *debugging* (depuración) con la opción "-X", y opcionalmente utilizando la opción "-C" para salida inmediata:

```
$ radiusd -X

Starting - reading configuration files ...
including dictionary file /etc/raddb/dictionary
radiusd: #### Loading Clients ####
client localhost {
ipaddr = 127.0.0.1
require_message_authenticator = no
secret = "testing123"
nastype = "other"
}
client 192.168.1.1 {
require_message_authenticator = no
secret = "testing123"
nastype = "other"
}
radiusd: #### Instantiating modules ####
```

```
radiusd: #### Loading Virtual Servers ####
Module: Linked to module rlm_eap
Module: Instantiating module "eap" from file /etc/raddb/eap.conf
eap {
default_eap_type = "tls"
timer_expire = 60
ignore_unknown_eap_types = no
cisco_accounting_username_bug = no
max_sessions = 4096
}
Module: Linked to sub-module rlm_eap_md5
Module: Instantiating eap-md5
Module: Linked to sub-module rlm_eap_tls
Module: Instantiating eap-tls
tls {
rsa_key_exchange = no
dh_key_exchange = yes
rsa_key_length = 512
dh_key_length = 512
verify_depth = 0
CA_path = "/etc/raddb/certs"
pem_file_type = yes
private_key_file = "/etc/raddb/certs/radkey.pem"
certificate_file = "/etc/raddb/certs/radcrt.pem"
CA_file = "/etc/raddb/certs/crlca.pem"
dh_file = "/etc/raddb/certs/dh"
random_file = "/etc/raddb/certs/random"
```

fragment_size = 1024

include_length = yes

check_crl = yes

cipher_list = "DEFAULT"

make_cert_command = "/etc/raddb/certs/bootstrap"

ecdh_curve = "prime256v1"

cache {

enable = no

lifetime = 24

max_entries = 255

}

verify {

}

ocsp {

enable = no

override_cert_url = yes

url = "http://127.0.0.1/ocsp/"

use_nonce = yes

timeout = 0

softfail = no

}

}

Module: Linked to module rlm_files

Module: Instantiating module "files" from file /etc/raddb/modules/files

files {

usersfile = "/etc/raddb/users"

}

reading pairlist file /etc/raddb/users

radiusd: #### Opening IP addresses and Ports ####

Listening on authentication address * port 1812

Listening on accounting address * port 1813

Ready to process requests.

El servidor lee los ficheros de configuración, habilita los clientes radius (NAS, AP), instancia los módulos (EAP, Files), asigna las direcciones y los puertos, y finalmente queda a la espera de solicitudes de acceso. Este último mensaje de espera es indicativo de que el servidor no manifiesta ningún problema de configuración, porque si fuese así el servidor terminaría inesperadamente mostrando detalles sobre el problema.

Conexión cableada con mecanismo EAP-MD5

Una vez conectado el conmutador se habilita el método *md5* en el fichero */etc/raddb/eap.conf,* se activa el usuario *bob* en el fichero */etc/raddb/users*, se habilita el módulo *files* en el fichero */etc/raddb/sites-enabled/default,* y se añade un nuevo bloque con la dirección del equipo origen y la *password* “testing” en el fichero */etc/raddb/clients.conf*. Ahora se puede proceder a probar el funcionamiento e integración del sub-sistema cliente-conmutador-servidor mediante un método simple de autenticación como EAP-MD5.

Es importante notar que aquí no existe autenticador, simplemente se está probando el enlace *end-to-end* con el servidor RADIUS, y poniendo a prueba el instanciamiento y uso de sus módulos.

El sub-sistema se encuentra en la sub-red 192.168.0.0/29. Concretamente el host RADIUS está dando servicio en la dirección 192.168.0.3, el conmutador en la 192.168.0.1, y el cliente (origen) en la 192.168.0.4. Para realizar el test se emplea la herramienta *radeapclient*, y para ello se ejecuta lo siguiente en el terminal:

```
$ cat << EOF | radeapclient -x 192.168.0.3 auth testing
User-Name = "bob"
Cleartext-Password = "hello"
EAP-Code = Response
EAP-Type-Identity = "bob"
EOF

Sending Access-Request packet to host 192.168.0.3 port 1812, id=123, length=0
User-Name = "bob"
Cleartext-Password = "hello"
EAP-Code = Response
EAP-Type-Identity = "bob"
EAP-Message = 0x027a000801626f62
Message-Authenticator = 0x00000000000000000000000000000000

Received Access-Challenge packet from host 192.168.0.3 port 1812, id=123, length=92
Reply-Message = "Hello, bob"
EAP-Message = 0x017b0016041015cacf99bffa31149d2e634727a51b65
Message-Authenticator = 0x43a2df0e82045079ca3e428eb5654d45
State = 0x7bd316557ba812adfca1fcd44b6cacd0
EAP-Id = 123
```

EAP-Code = Request

EAP-Type-MD5 = 0x1015cacf99bffa31149d2e634727a51b65

Sending Access-Request packet to host 192.168.0.3 port 1812, id=124, length=53

User-Name = "bob"

Cleartext-Password = "hello"

EAP-Code = Response

Message-Authenticator = 0x00000000000000000000000000000000

EAP-Type-MD5 = 0x10e949847f2f6a61e1330859c864ed23f8

EAP-Id = 123

State = 0x7bd316557ba812adfca1fcd44b6cacd0

EAP-Message = 0x027b00160410e949847f2f6a61e1330859c864ed23f8

Received Access-Accept packet from host 192.168.0.3 port 1812, id=124, length=61

Reply-Message = "Hello, bob"

EAP-Message = 0x037b0004

Message-Authenticator = 0x9ae0a5d0586f9da7c7222d5b5455c574

User-Name = "bob"

EAP-Id = 123

EAP-Code = Success

La conexión se ha realizado con éxito al recibir un mensaje *Access-Accept* desde el servidor RADIUS a través del *switch,* con EAP-Code *Success.* Como se puede apreciar el cliente envía una *request* (solicitud de acceso) que es respondida con un *challenge-md5,* a lo cual el cliente contesta adecuadamente y se establece la conexión.

En el supuesto caso de que exista algún problema con el servidor RADIUS y este no responda, habrá que conectarse al *switch* y probar el *hop-to-hop* entre servidor y conmutador, o incluso entre cliente y *switch*.

Acceso cableado/inalámbrico con método EAP-TLS

Para este caso de uso se pueden alternar manualmente las configuraciones de ambos entornos, cableado e inalámbrico, descritas en los apartados 5.3 y 5.4 (Despliegue de acceso cableado y Anexión de acceso inalámbrico), porque no existe distinción alguna entre los dos tipos de acceso desde el punto de vista de las acciones llevadas a cabo por el suplicante para realizar el acoplamiento, pues ambos soportes (cableado e inalámbrico) están implementados en el *kernel* del sistema operativo, y el suplicante hace uso de ellos.

En primer lugar y si se acaba de realizar el caso de uso anterior se deben deshabilitar las configuraciones específicas que se establecieron. Posteriormente y habiendo configurado uno de los dos entornos (cableado/inalámbrico) se ejecuta el suplicante según lo detallado en el apartado correspondiente, y se obtiene como resultado la información de traza para la conexión del empleado a la infraestructura corporativa.

Para este escenario se considera que el empleado posee credenciales vigentes y está ubicado en el interior/exterior de la sede corporativa:

```
$ wpa_supplicant -i <iface> -D <driver> -c/etc/wpa_supplicant.conf

rl0: CTRL-EVENT-EAP-STARTED EAP authentication started
```

rl0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=13

rl0: CTRL-EVENT-EAP-METHOD EAP vendor 0 method 13 (TLS) selected

rl0: CTRL-EVENT-EAP-PEER-CERT depth=1 subject='/C=ES/O=SisCE OU=SecRootCA/emailAddress=sec@sisce.esc'

rl0: CTRL-EVENT-EAP-PEER-CERT depth=0 subject='/C=ES/O=SisCE OU=SecAUTH/CN=rad.sisce.esc'

rl0: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully

Estas líneas indican al empleado que la negociación EAP (capa 2) ha comenzado y que el método ofrecido por su suplicante al servidor de autenticación es EAP-TLS (identificador 13), el cual ha sido aceptado por este último. A continuación se verifican los certificados PKI de autoridad (CA) y de servidor (Radius), donde se puede apreciar que la jerarquía de certificación tiene 2 niveles (depth=1 de la CA), y finalmente el éxito del acoplamiento a la infraestructura es notificado al empleado por el suplicante.

El empleo de una tarjeta de identificación electrónica haciendo uso del interfaz de gestión del suplicante es total y visualmente transparente al empleado, siendo este uno de los objetivos del sistema, la transparencia.

Ahora puede probarse la conectividad *end-to-end* con el sevidor mediante el protocolo ICMP:

$ ping –c2 192.168.0.3

PING rad.sisce.esc (192.168.0.3) 56(84) bytes of data.

64 bytes from rad.sisce.esc (192.168.0.3): icmp_seq=1 ttl=255 time=0.122 ms

64 bytes from rad.sisce.esc (192.168.0.3): icmp_seq=2 ttl=255 time=0.186 ms

--- 192.168.0.3 ping statistics ---

```
2 packets transmitted, 2 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.179/0.257/0.402/0.103 ms
```

Esta conectividad es correcta tras la conexión, pero si se intenta realizar antes del establecimiento se observará que el servidor RADIUS no responde, porque el autenticador sólo permite tráfico de paquetes EAPoL/W (paquetes EAP) hacia el servidor AAA.

Si el empleado utiliza un certificado personal expirado, revocado, y/o firmado por una autoridad no confiable, se visualiza la siguiente traza:

```
rl0: CTRL-EVENT-DISCONNECTED reason=3 locally_generated=1
```

Esto indica al empleado que la elección del certificado local es errónea (reason=3).

Verificar configuración del servidor VPN

Tras la ejecución de los comandos del apartado 5.5 (Anexión de acceso remoto) para configurar el servicio de conexión remota, se comprueba la correcta configuración de este último ejecutando el *daemon* de OpenVPN, y opcionalmente si se observa alguna anomalía se puede ejecutar en modo *debugging* con el parámetro “--verb 6”:

```
$ openvpn --config /etc/openvpn/server.conf

Fri Jun 20 15:19:25 2014 Diffie-Hellman initialized with 1024 bit key
Fri Jun 20 15:19:25 2014 Socket Buffers: R=[41600->65536] S=[9216->65536]
Fri Jun 20 15:19:25 2014 TUN/TAP device /dev/tun0 opened
```

Fri Jun 20 15:19:25 2014 do_ifconfig, tt->ipv6=0, tt->did_ifconfig_ipv6_setup=0

Fri Jun 20 15:19:25 2014 /sbin/ifconfig tun0 10.8.0.1 10.8.0.2 mtu 1500 netmask 255.255.255.255 up -link0

Fri Jun 20 15:19:25 2014 /sbin/route add -net 10.8.0.0 10.8.0.2 -netmask 255.255.255.0

add net 10.8.0.0: gateway 10.8.0.2

Fri Jun 20 15:19:25 2014 GID set to _openvpn

Fri Jun 20 15:19:25 2014 UID set to _openvpn

Fri Jun 20 15:19:25 2014 UDPv4 link local (bound): [undef]

Fri Jun 20 15:19:25 2014 UDPv4 link remote: [undef]

Fri Jun 20 15:19:25 2014 MULTI: multi_init called, r=256 v=256

Fri Jun 20 15:19:25 2014 IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0

Fri Jun 20 15:19:25 2014 IFCONFIG POOL LIST

Fri Jun 20 15:19:25 2014 Initialization Sequence Completed

El servidor muestra la secuencia de inicialización, creando el interfaz virtual (tun0), asignando direcciones, añadiendo rutas, y delegando privilegios y permisos, para finalmente mostrar un indicador de inicialización completada, quedando a la espera de conexiones entrantes.

Si algo fuese mal durante la inicialización del servicio el servidor terminaría inesperadamente mostrando algún indicativo de error, y tras ello sería conveniente proceder con el método de depuración.

Acceso remoto con método UDP-TLS

Una vez realizada la configuración descrita en el apartado 5.5 (Anexión de acceso remoto) se puede iniciar la conexión haciendo uso del cliente que ofrece la *suite* OpenVPN.

Para este escenario se considera un acceso remoto a la infraestructura desde la perspectiva de un empleado con credenciales vigentes y ubicado en su residencia familiar:

```
$ openvpn --config /etc/openvpn/client.conf
```

```
Fri Jun  6 09:10:00 2014 VERIFY OK: depth=1, C=ES, O=SisCE, OU=SecRootCA,
emailAddress=sec@sisce.esc

Fri Jun  6 09:10:00 2014 VERIFY OK: nsCertType=SERVER

Fri Jun  6 09:10:00 2014 VERIFY OK: depth=0, C=ES, O=SisCE, OU=SecVPN,
CN=tun.sisce.esc

Fri Jun  6 09:10:00 2014 Data Channel Encrypt: Cipher 'BF-CBC' initialized with 128 bit
key

Fri Jun  6 09:10:00 2014 Data Channel Encrypt: Using 160 bit message hash 'SHA1'
for HMAC authentication

Fri Jun  6 09:10:00 2014 Data Channel Decrypt: Cipher 'BF-CBC' initialized with 128 bit
key

Fri Jun  6 09:10:00 2014 Data Channel Decrypt: Using 160 bit message hash 'SHA1'
for HMAC authentication

Fri Jun  6 09:10:00 2014 Control Channel: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-
AES256-SHA, 2048 bit RSA

Fri Jun  6 09:10:00 2014 [tun.sisce.esc] Peer Connection Initiated with
[AF_INET]192.168.0.3:1194

Fri Jun  6 09:10:02 2014 SENT CONTROL [tun.sisce.esc]: 'PUSH_REQUEST'
(status=1)

Fri Jun  6 09:10:02 2014 PUSH: Received control message:'PUSH_REPLY,route
10.8.0.1,topology net30,ping 10,ping-restart 120,ifconfig 10.8.0.6 10.8.0.5'
```

Fri Jun 6 09:10:02 2014 ROUTE_GATEWAY 192.168.0.1

Fri Jun 6 09:10:02 2014 TUN/TAP device /dev/tun0 opened

Fri Jun 6 09:10:02 2014 /sbin/ifconfig tun0 10.8.0.6 10.8.0.5 mtu 1500 netmask 255.255.255.255 up

Fri Jun 6 09:10:02 2014 /sbin/route add -net 192.168.1.0 10.8.0.5 255.255.255.0 add net 192.168.1.0: gateway 10.8.0.5

Fri Jun 6 09:10:02 2014 /sbin/route add -net 10.8.0.1 10.8.0.5 255.255.255.255 add net 10.8.0.1: gateway 10.8.0.5

Fri Jun 6 09:10:02 2014 GID set to nobody

Fri Jun 6 09:10:02 2014 UID set to nobody

Fri Jun 6 09:10:02 2014 Initialization Sequence Completed

En primer lugar se verifica el certificado de la autoridad (CA) y el del servidor (VPN), donde se comprueba que el certificado de servidor es de uso exclusivo para servidores (nsCertType). A continuación se declaran parámetros para el establecimiento del túnel cifrado (BF-CBC, SHA1, RSA 2048), y se notifica el establecimiento del enlace con el *peer* (en este caso el servidor). Ahora el servidor envía al cliente los parámetros de configuración del sistema (PUSH_REPLY), que son aplicados sobre la tabla de rutas y el interfaz de tunelización (tun0) mediante los comandos *ifconfig* y *route*. Tras esto se minimizan los privilegios del *daemon* y se notifica al empleado que se puede iniciar el tráfico de datos de la sesión de usuario.

En su configuración por defecto OpenVPN utiliza la sub-red privada 10.8.0.0/24 y el servidor toma la IP 10.8.0.1, por consiguiente para probar el enlace es suficiente con enviar algunos paquetes ICMP a esa dirección y observar que se reciben correctamente las respuestas:

```
$ ping –c2 10.8.0.1
```

```
PING tun.sisce.esc (10.8.0.1) 56(84) bytes of data.
64 bytes from tun.sisce.esc (10.8.0.1): icmp_seq=1 ttl=64 time=0.499 ms
64 bytes from tun.sisce.esc (10.8.0.1): icmp_seq=2 ttl=64 time=0.200 ms
--- 10.8.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.200/0.349/0.499/0.150 ms
```

Si el empleado utiliza un certificado expirado, revocado, y/o firmado por una autoridad no confiable, se visualiza la siguiente traza de error:

```
Fri Jun  6 13:44:20 2014 192.168.0.4:35523 TLS Error: TLS handshake failed
```

A partir de esta información el empleado deduce que el servidor ha notificado un error durante la fase de negociación del protocolo TLS, lo cual probablemente es debido al empleo de un certificado erróneo.

Verificar la conexión a la LAN mediante ICMP

Para poder verificar que se alcanza la red interna de la organización tras una conexión exitosa, es necesario previamente añadir las rutas adecuadas a los dispositivos encaminadores, para que los paquetes enviados puedan alcanzar la red interna (remota/destino), pero igualmente las respuestas puedan ser devueltas al cliente convenientemente (origen), como se explicó en los apartados del capítulo 5 (Realización del proyecto).

Si se supone una LAN ubicada en el segmento de red 192.168.1.0/24, es suficiente con efectuar un *ping* hacia un *host* existente y activo en ese segmento, tras haber establecido previamente una conexión satisfactoria

con la infraestructura de red a través de cualquiera de los métodos de conexión habilitados por la organización (cableado, inalámbrico, o remoto):

```
$ ping –c2 192.168.1.40

PING web.sisce.esc (192.168.1.40) 56(84) bytes of data.
64 bytes from web.sisce.esc (192.168.1.40): icmp_seq=1 ttl=255 time=0.139 ms
64 bytes from web.sisce.esc (192.168.1.40): icmp_seq=2 ttl=255 time=0.190 ms
--- 192.168.1.40 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.195/0.264/0.397/0.094 ms
```

El resultado confirma el éxito de la conectividad entre el cliente (empleado) y la *intranet* corporativa.

7. CONCLUSIONES Y PERSPECTIVAS DE FUTURO

7. Conclusiones y perspectivas de futuro

Durante la realización de este proyecto se ha podido comprobar que el sistema desplegado es muy robusto y seguro, pudiendo alcanzar distintos niveles de seguridad en función de los requisitos y de las necesidades corporativas.

Si se aplica una infraestructura PKI completa y se cumplen los procedimientos (normativas y regulaciones) actualmente definidos en los estándares de seguridad, el único punto débil aparente lo impone el propio desarrollo tecnológico, pues como es bien sabido las claves criptográficas son vulnerables al incremento progresivo de la capacidad computacional. Un sistema es tan seguro como el estado del arte de la tecnología se lo permite y su administrador quiere que lo sea.

La gran mayoría de los problemas de seguridad que se producen en los sistemas se deben a un mantenimiento desactualizado, lo cual está estrechamente relacionado con la falta de auditorías periódicas. Desde aquellos tiempos en los que una conexión PPP se consideraba segura por el mero hecho de ser un enlace punto a punto a través de una línea serie ha pasado mucho tiempo, y la criptografía ha inundado los sistemas, incrementando su complejidad y soportando todo el peso de la seguridad de los mismos.

El siguiente paso es probablemente una fusión de dispositivos criptográficos, y en esta línea el teléfono inteligente es el candidato idóneo para soportarlo todo. No sería algo nuevo ni extraño imaginar sistemas de control basados en tecnología criptográfica soportada en los actuales *smart*

phones, integrando credenciales biométricos o reconocedores de los mismos, y permitiendo el acceso a cualquier tipo de sistema, desde el propio domicilio familiar pasando por la compra de artículos o el lugar de trabajo hasta el mismísimo estado de salud de su propietario (“u otros”).

El mercado del entretenimiento ya comienza a mostrar al consumidor algunos prototipos de realidad virtual aumentada, y el campo sensorial y de la percepción cognitiva está siendo aplicado a distintas áreas tecnológicas, tales como los teléfonos inteligentes. Realmente parece ciencia ficción, pero estamos muy próximos a formar parte de una sociedad en la que lo biológico esté tan íntimamente relacionado con lo artificial que puedan atacarse mutuamente, y la única barrera posible sea la correcta aplicación de métodos criptográficos seguros y eficientes.

Todo esto abre las puertas a un sin fin de posibilidades, cuya constante máxima pretensión es mejorar la vida de los usuarios, y buscar los medios para permitir el avance sin poner en riesgo la seguridad de la información gestionada. Pero en la mayor parte de los casos se piensa primero en el avance tecnológico, y después en su seguridad, propiciando el porcentaje restante a un mantenimiento desactualizado.

El conocimiento ya se posee, tan sólo hay que valorar debidamente sus prioridades y su orden de aplicación.

8. BIBLIOGRAFÍA

8. Bibliografía

Enlaces:

http://hostap.epitest.fi/wpa_supplicant

http://freeradius.org

http://deployingradius.com

https://www.openssl.org

http://www.rfc-archive.org

https://github.com/OpenSC/OpenSC/wiki

http://openvpn.net/index.php/open-source.html

http://www.freebsd.org/doc/handbook

Libros:

RADIUS / AAA / 802.1X. Sistemas basados en la autenticación.
(Yago Fernández Hansen, Antonio Ramos Varón, Jean Paul García-Morán)

TCP/IP Illustrated, Volume1. The Protocols.
(Kevin R. Fall, W. Richard Stevens)

Secure Architectures with OpenBSD.
(Brandon Palmer, Jose Nazario)

9. APÉNDICE

9. Apéndice

En este capítulo final se anexan todos los ficheros de configuración que han sufrido modificaciones durante la realización del proyecto, además de los de nueva creación. Igualmente y debido a la extensión, se han suprimido todos los comentarios contenidos en los ficheros.

La siguiente lista contiene los archivos relevantes para el correcto funcionamiento del sistema implementado:

- /etc/raddb/radiusd.conf
- /etc/raddb/users
- /etc/raddb/sites-enabled/default
- /etc/raddb/eap.conf
- /etc/raddb/clients.conf
- /etc/openvpn/client.conf
- /etc/openvpn/server.conf
- /usr/local/etc/openssl/openssl.cnf
- /etc/wpa_supplicant.conf

Fichero - /etc/raddb/radiusd.conf

```
prefix = /usr/local

exec_prefix = ${prefix}

sysconfdir = /etc

localstatedir = /var

sbindir = ${exec_prefix}/sbin

logdir = ${localstatedir}/log/radius

raddbdir = ${sysconfdir}/raddb

radacctdir = ${logdir}/radacct

name = radiusd

confdir = ${raddbdir}

run_dir = ${localstatedir}/run/${name}

db_dir = ${raddbdir}

libdir = /usr/local/lib/freeradius

pidfile = ${run_dir}/${name}.pid

# chroot = /path/to/chroot/directory

user = _freeradius

group = _freeradius

max_request_time = 30

cleanup_delay = 5

max_requests = 1024

listen {

      type = auth

      ipaddr = 192.168.0.3

#     ipv6addr = ::    # any.  ::1 == localhost
```

```
        port = 1812
#       interface = eth0
#       clients = per_socket_clients
}
listen {
        ipaddr = *
#       ipv6addr = ::
        port = 0
        type = acct
#       interface = eth0
#       clients = per_socket_clients
}
hostname_lookups = no
allow_core_dumps = no
regular_expressions     = yes
extended_expressions    = yes
log {
        destination = files
        file = ${logdir}/radius.log
        syslog_facility = daemon
        stripped_names = no
        auth = yes
        auth_badpass = yes
        auth_goodpass = yes
#       msg_goodpass = ""
#       msg_badpass = ""
```

```
}
checkrad = ${sbindir}/checkrad
security {
	max_attributes = 200
	reject_delay = 1
	status_server = yes
}
proxy_requests  = no
# $INCLUDE ${confdir}/proxy.conf
$INCLUDE ${confdir}/clients.conf
thread pool {
	start_servers = 5
	max_servers = 32
	min_spare_servers = 3
	max_spare_servers = 10
#	max_queue_size = 65536
	max_requests_per_server = 0
}
modules {
	$INCLUDE ${confdir}/modules/
	$INCLUDE ${confdir}/eap.conf
#	$INCLUDE ${confdir}/sql.conf
#	$INCLUDE ${confdir}/sql/mysql/counter.conf
#	$INCLUDE ${confdir}/sqlippool.conf
}
instantiate {
```

```
        exec
        expr
#       daily
        expiration
        logintime
}
$INCLUDE ${confdir}/policy.conf
$INCLUDE sites-enabled/
```

Fichero - /etc/raddb/users

```
# lameuser  Auth-Type := Reject
#       Reply-Message = "Your account has been disabled."
# DEFAULT Group == "disabled", Auth-Type := Reject
#       Reply-Message = "Your account has been disabled."
Steve Cleartext-Password := "testing"
        Service-Type = Framed-User,
        Framed-Protocol = PPP,
        Framed-IP-Address = 172.16.3.33,
        Framed-IP-Netmask = 255.255.255.0,
        Framed-Routing = Broadcast-Listen,
        Framed-Filter-Id = "std.ppp",
        Framed-MTU = 1500,
        Framed-Compression = Van-Jacobsen-TCP-IP
bob Cleartext-Password := "hello"
        Reply-Message = "Hello, %{User-Name}"
```

```
# "John Doe"         Cleartext-Password := "hello"
#    Reply-Message = "Hello, %{User-Name}"
# DegCleartext-Password := "ge55ged"
#    Service-Type = Callback-Login-User,
#    Login-IP-Host = 0.0.0.0,
#    Callback-Number = "9,5551212",
#    Login-Service = Telnet,
#    Login-TCP-Port = Telnet
# dialbk       Cleartext-Password := "callme"
#    Service-Type = Callback-Login-User,
#    Login-IP-Host = timeshare1,
#    Login-Service = PortMaster,
#    Callback-Number = "9,1-800-555-1212"
# swilson     Service-Type == Framed-User, Huntgroup-Name == "alphen"
#    Framed-IP-Address = 192.168.1.65,
#    Fall-Through = Yes
# DEFAULT Suffix == ".shell"
#    Service-Type = Login-User,
#    Login-Service = Telnet,
#    Login-IP-Host = your.shell.machine
# DEFAULT Service-Type == Framed-User, Huntgroup-Name == "alphen"
#    Framed-IP-Address = 192.168.1.32+,
#    Fall-Through = Yes
# DEFAULT Service-Type == Framed-User, Huntgroup-Name == "delft"
#    Framed-IP-Address = 192.168.2.32+,
#    Fall-Through = Yes
```

```
# DEFAULT Service-Type == Framed-User
#    Framed-IP-Address = 255.255.255.254,
#    Framed-MTU = 576,
#    Service-Type = Framed-User,
#    Fall-Through = Yes
DEFAULT    Framed-Protocol == PPP
     Framed-Protocol = PPP,
     Framed-Compression = Van-Jacobson-TCP-IP
DEFAULT    Hint == "CSLIP"
     Framed-Protocol = SLIP,
     Framed-Compression = Van-Jacobson-TCP-IP
DEFAULT    Hint == "SLIP"
     Framed-Protocol = SLIP
# DEFAULT
#    Service-Type = Login-User,
#    Login-Service = Rlogin,
#    Login-IP-Host = shellbox.ispdomain.com
# DEFAULT
#    Service-Type = Administrative-User
```

Fichero - /etc/raddb/sites-enabled/default

```
authorize {
#    filter_username
     preprocess
#    auth_log
#    chap
#    mschap
#    digest
#    wimax
#    IPASS
#    suffix
#    ntdomain
#    update control {
#        EAP-TLS-Require-Client-Cert = Yes
#    }
     eap {
        ok = return
     }
#    unix
     files
#    sql
#    etc_smbpasswd
#    ldap
#    daily
#    checkval
```

```
#   expiration
#   logintime
#   pap
#   Autz-Type Status-Server {
#   }
}
authenticate {
    Auth-Type PAP {
        pap
    }
    Auth-Type CHAP {
        chap
    }
    Auth-Type MS-CHAP {
        mschap
    }
    digest
#   pam
    unix
#   Auth-Type LDAP {
#       ldap
#   }
    eap
#   Auth-Type eap {
#       eap {
#           handled = 1
```

```
#       }
#       if (handled && (Response-Packet-Type == Access-Challenge)) {
#           attr_filter.access_challenge.post-auth
#           handled  # override the "updated" code from attr_filter
#       }
#   }
}
preacct {
    preprocess
#   update request {
#   FreeRADIUS-Acct-Session-Start-Time = "%{expr: %l - %{%{Acct-
    Session-Time}:-0} - %{%{Acct-Delay-Time}:-0}}"
#   }
    acct_unique
#   IPASS
    suffix
#   ntdomain
    files
}
accounting {
    detail
#   daily
#   unix
#   radutmp
#   sradutmp
#   main_pool
```

```
#       sql
#       if (noop) {
#           ok
#       }
#       sql_log
#       pgsql-voip
        exec
        attr_filter.accounting_response
#       Acct-Type Status-Server {
#       }
}
session {
        radutmp
#       sql
}
post-auth {
#       main_pool
#       reply_log
#       sql
#       sql_log
#       ldap
        exec
#       wimax
#       update reply {
#       Reply-Message += "%{TLS-Cert-Serial}"
#       Reply-Message += "%{TLS-Cert-Expiration}"
```

```
#   Reply-Message += "%{TLS-Cert-Subject}"
#   Reply-Message += "%{TLS-Cert-Issuer}"
#   Reply-Message += "%{TLS-Cert-Common-Name}"
#   Reply-Message += "%{TLS-Cert-Subject-Alt-Name-Email}"
#   Reply-Message += "%{TLS-Client-Cert-Serial}"
#   Reply-Message += "%{TLS-Client-Cert-Expiration}"
#   Reply-Message += "%{TLS-Client-Cert-Subject}"
#   Reply-Message += "%{TLS-Client-Cert-Issuer}"
#   Reply-Message += "%{TLS-Client-Cert-Common-Name}"
#   Reply-Message += "%{TLS-Client-Cert-Subject-Alt-Name-Email}"
#   }
    Post-Auth-Type REJECT {
#       log failed authentications in SQL, too.
#       sql
        attr_filter.access_reject
    }
}
pre-proxy {
#   attr_rewrite
#   files
#   attr_filter.pre-proxy
#   pre_proxy_log
}
post-proxy {
#   post_proxy_log
#   attr_rewrite
```

```
#   attr_filter.post-proxy
    eap
#   Post-Proxy-Type Fail {
#       detail
#   }
}
```

Fichero - /etc/raddb/eap.conf

```
eap {
    default_eap_type = md5
    # default_eap_type = tls
    timer_expire     = 60
    ignore_unknown_eap_types = no
    cisco_accounting_username_bug = no
    max_sessions = 4096
    md5 {
    }
    leap {
    }
    gtc {
#       challenge = "Password: "
        auth_type = PAP
    }
    tls {
        certdir = ${confdir}/certs
```

```
        cadir = ${confdir}/certs

        private_key_file = ${certdir}/radkey.pem

        certificate_file = ${certdir}/radcrt.pem

        CA_file = ${cadir}/crlca.pem

        dh_file = ${certdir}/dh

        random_file = ${certdir}/random

#       fragment_size = 1024

#       include_length = yes

        check_crl = yes

        CA_path = ${cadir}

#       check_cert_issuer = "/C=GB/ST=Berkshire/L=Newbury/O=My
        Company Ltd"

#       check_cert_cn = %{User-Name}

        cipher_list = "DEFAULT"

        make_cert_command = "${certdir}/bootstrap"

        ecdh_curve = "prime256v1"

        cache {

            enable = no

            lifetime = 24 # hours

            max_entries = 255

        }

        verify {

#           tmpdir = /tmp/radiusd

#           client = "/path/to/openssl verify -CApath ${..CA_path}
            %{TLS-Client-Cert-Filename}"

        }

        ocsp {
```

```
            enable = no
            override_cert_url = yes
            url = "http://127.0.0.1/ocsp/"
#           use_nonce = yes
#           timeout = 0
#           softfail = no
        }
    }
    ttls {
        default_eap_type = md5
        copy_request_to_tunnel = no
        use_tunneled_reply = no
        virtual_server = "inner-tunnel"
#       include_length = yes
    }
    peap {
        default_eap_type = mschapv2
        copy_request_to_tunnel = no
        use_tunneled_reply = no
#       proxy_tunneled_request_as_eap = yes
        virtual_server = "inner-tunnel"
#       soh = yes
#       soh_virtual_server = "soh-server"
    }
    mschapv2 {
#       send_error = no
```

```
    }
}
```

Fichero - /etc/raddb/clients.conf

```
client localhost {
    ipaddr = 127.0.0.1
#   ipv6addr = ::  # any.  ::1 == localhost
#   netmask = 32
    secret = testing123
    require_message_authenticator = no
#   shortname = localhost
    nastype = other                 # localhost isn't usually a NAS...
#   login = !root
#   password = someadminpas
#   virtual_server = home1
#   coa_server = coa
}
# client ::1 {
#   secret = testing123
#   shortname = localhost
# }
# client fe80::/16 {
#   secret = testing123
#   shortname = localhost
# }
```

```
# client some.host.org {
#    secret = testing123
#    shortname = localhost
# }
# client 192.168.0.0/24 {
#    secret = testing123-1
#    shortname = private-network-1
# }
# client 192.168.0.0/16 {
#    secret          = testing123-2
#    shortname       = private-network-2
#}
# client 10.10.10.10 {
#    secret = testing123
#    shortname = liv1
#    nastype = livingston
#    login = !root
#    password = someadminpas
# }
# clients per_socket_clients {
#    client 192.168.3.4 {
#        secret = testing123
#    }
# }
client 192.168.0.1 {
     secret = testing123
```

```
    nastype = other
}
client 192.168.0.4 {
    secret = testing
    nastype = other
}
```

Fichero - /etc/openvpn/client.conf

```
client
; dev tap
dev tun
; dev-node MyTap
; proto tcp
proto udp
remote tun.sisce.esc 1194
; remote my-server-2 1194
; remote-random
resolv-retry infinite
nobind
user nobody
group nogroup
persist-key
persist-tun
; http-proxy-retry # retry on connection failures
; http-proxy [proxy server] [proxy port #]
```

```
; mute-replay-warnings

ca /etc/openvpn/cacrt.pem

# cert /etc/openvpn/clncrt.pem

# key /etc/openvpn/clnkey.pem

management localhost 7505

management-query-passwords

pkcs11-providers /usr/local/lib/opensc-pkcs11.so

pkcs11-id
'EnterSafe/PKCS\x2315/1320463413190613/OpenSC\x20Card\x20\x28
User\x20PIN\x29/31'

ns-cert-type server

; tls-auth ta.key 1

cipher BF-CBC

comp-lzo

verb 3

; mute 20
```

Fichero - /etc/openvpn/server.conf

```
local 192.168.0.2
port 1194
proto udp
; dev tap
dev tun
ca /etc/openvpn/cacrt.pem
cert  /etc/openvpn/tuncrt.pem
key  /etc/openvpn/tunkey.pem    # This file should be kept secret
crl-verify  /etc/openvpn/crl.pem
dh  /etc/openvpn/dh
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist /etc/openvpn/ipp.txt
; server-bridge 10.8.0.4 255.255.255.0 10.8.0.50 10.8.0.100
; server-bridge
; push "route 192.168.10.0 255.255.255.0"
; push "route 192.168.20.0 255.255.255.0"
push "route 192.168.1.0 255.255.255.0"
; client-config-dir ccd
; route 192.168.40.128 255.255.255.248
; client-config-dir ccd
; route 10.9.0.0 255.255.255.252
; learn-address ./script
; push "redirect-gateway def1 bypass-dhcp"
; push "redirect-gateway def1"
```

```
; push "dhcp-option DNS 208.67.222.222"
; push "dhcp-option DNS 208.67.220.220"
; client-to-client
; duplicate-cn
keepalive 10 120
; tls-auth ta.key 0                # This file is secret
cipher BF-CBC                      # Blowfish (default)
; cipher AES-128-CBC               # AES
; cipher DES-EDE3-CBC              # Triple-DES
comp-lzo
; max-clients 100
user _openvpn
group _openvpn
persist-key
persist-tun
status  openvpn-status.log
; log  openvpn.log
; log-append  openvpn.log
verb 3
;mute 20
```

Fichero - /usr/local/etc/openssl/openssl.cnf

```
HOME = .
RANDFILE = $ENV::HOME/.rnd
# oid_file = $ENV::HOME/.oid
oid_section = new_oids
[ new_oids ]
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7
[ ca ]
default_ca = CA_default                  # The default ca section
[ CA_default ]
dir = .                                 # Where everything is kept
certs = $dir/certs                      # Where the issued certs are kept
crl_dir = $dir/crl                      # Where the issued crl are kept
database = $dir/index                   # database index file.
# unique_subject = no                   # Set to 'no' to allow creation of
new_certs_dir         = $dir/store      # default place for new certs.
certificate = $dir/cacrt.pem            # The CA certificate
serial = $dir/serial                    # The current serial number
# crlnumber = $dir/number               # the current crl number
crl = $dir/crl.pem                      # The current CRL
private_key = $dir/cakey.pem            # The private key
RANDFILE = $dir/.rand                   # private random number file
x509_extensions = usr_cert              # The extentions to add to the cert
```

```
name_opt = ca_default                # Subject Name options
cert_opt = ca_default                # Certificate field options
# crl_extensions = crl_ext
default_days = 730                   # how long to certify for
default_crl_days = 30                # how long before next CRL
default_md = default                 # use public key default MD
preserve = no                        # keep passed DN ordering
policy = policy_match
[ policy_match ]
countryName = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
[ policy_anything ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
[ req ]
default_bits = 2048
default_keyfile = privkey.pem
```

```
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca
# input_password = secret
# output_password = secret
string_mask = utf8only
# req_extensions = v3_req
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = es
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = madrid
localityName = Locality Name (eg, city)
0.organizationName = Organization Name (eg, company)
0.organizationName_default = mi-empresa-web
# 1.organizationName = Second Organization Name (eg, company)
# 1.organizationName_default = World Wide Web Pty Ltd
organizationalUnitName = Organizational Unit Name (eg, section)
# organizationalUnitName_default =
commonName = Common Name (e.g. server FQDN or YOUR name)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64
# SET-ex3 = SET extension number 3
```

```
[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20
unstructuredName = An optional company name
[ usr_cert ]
basicConstraints = CA:FALSE
# nsCertType = server
# nsCertType = objsign
# nsCertType = client, email
# nsCertType = client, email, objsign
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
# subjectAltName = email:copy
# subjectAltName = email:move
# issuerAltName=issuer:copy
#nsCaRevocationUrl = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName
# extendedKeyUsage = critical,timeStamping
[ v3_req ]
```

```
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
[ v3_ca ]
basicConstraints = critical, CA:true, pathlen:0
# basicConstraints = CA:true
keyUsage = cRLSign, keyCertSign
subjectKeyIdentifier = hash
# authorityKeyIdentifier=keyid:always,issuer
# nsCertType = sslCA, emailCA
 subjectAltName = email:copy
# issuerAltName = issuer:copy
# obj=DER:02:03
# basicConstraints = critical, DER:30:03:01:01:FF
nsComment = "OpenSSL Generated Certificate"
[ client ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid
[ server ]
basicConstraints = CA:FALSE
# keyUsage = digitalSignature, keyEncipherment
keyUsage = digitalSignature, keyAgreement
extendedKeyUsage = serverAuth
```

```
nsCertType = server
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid
[ crl_ext ]
# issuerAltName = issuer:copy
authorityKeyIdentifier = keyid:always
[ proxy_cert_ext ]
basicConstraints = CA:FALSE
# nsCertType = server
# nsCertType = objsign
# nsCertType = client, email
# nsCertType = client, email, objsign
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
# subjectAltName = email:copy
# subjectAltName = email:move
# issuerAltName = issuer:copy
# nsCaRevocationUrl = http://www.domain.dom/ca-crl.pem
# nsBaseUrl
# nsRevocationUrl
# nsRenewalUrl
# nsCaPolicyUrl
# nsSslServerName
```

```
[ tsa ]

default_tsa = tsa_config1

[ tsa_config1 ]

dir = ./demoCA              # TSA root directory

serial = $dir/tsaserial     # The current serial number (mandatory)

crypto_device = builtin

signer_cert = $dir/tsacert.pem

certs = $dir/cacert.pem

signer_key = $dir/private/tsakey.pem

default_policy = tsa_policy1

other_policies = tsa_policy2, tsa_policy3

digests = md5, sha1

accuracy = secs:1, millisecs:500, microsecs:100

clock_precision_digits = 0

ordering = yes

tsa_name = yes

ess_cert_id_chain = no
```

Fichero - /etc/wpa_supplicant.conf

```
ctrl_interface=/var/run/wpa_supplicant

ctrl_interface_group=wheel

pkcs11_engine_path=/usr/local/lib/engines/engine_pkcs11.so

pkcs11_module_path=/usr/local/lib/opensc-pkcs11.so

eapol_version=

network={

ssid="one"

# key_mgmt=IEEE8021X

key_mgmt=WPA-EAP

eap=TLS

identity="SisCE-Worker"

# ca_cert="/home/ignacio/ca/cacrt.pem"

# client_cert="/home/ignacio/ca/clncrt.pem"

# private_key="/home/ignacio/ca/clnkey.pem"

engine=1

engine_id="pkcs11"

key_id="slot_1-id_51f191e2aa08d518a5f91620e7562a32a936545e"

cert_id="slot_1-id_51f191e2aa08d518a5f91620e7562a32a936545e"

ca_cert_id="slot_1-id_of37e5fca6e04e356d41b0193d725eb0e54fc3bd"

# pin="1234"

}
```